NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A DESIGN COMPARISON BETWEEN IPV4 AND IPV6 IN THE CONTEXT OF MYSEA, AND IMPLEMENTATION OF AN IPV6 MYSEA PROTOTYPE

by

Matthew R. O'Neal

June 2003

Thesis Advisor:	Cynthia E. Irvine
Second Reader:	Thuy D. Nguyen

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2002	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: A Design Comparison between IPv4 and IPv6 in the Context of MYSEA, and Implementation of an IPv6 MYSEA Prototype			**5. FUNDING NUMBERS**
6. AUTHOR(S) Matthew R. O'Neal			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

Internet Protocol version six (IPv6), the next generation Internet Protocol, exists sparsely in today's world. However, as it gains popularity, it will grow into a vital part of the Internet and communications technology in general. Many large organizations, including the Department of Defense, are working toward deploying IPv6 in many varied applications.

This thesis focuses on the design and implementation issues that accompany a migration from Internet Protocol version four (IPv4) to IPv6 in the Monterey Security Enhanced Architecture (MYSEA). The research for this thesis consists of two major parts: a functional comparison between the IPv6 and IPv4 designs, and a prototype implementation of MYSEA with IPv6.

The current MYSEA prototype relies on a subset of Network Address Translation (NAT) functionality to support the network's operation; and, due to the fact that IPv6 has no native support for NAT, this work also requires the creation of a similar mechanism for IPv6.

This thesis provides a preliminary examination of IPv6 in MYSEA, which is a necessary step in determining whether the new protocol will assist with or detract from the enforcement of MYSEA policies.

14. SUBJECT TERMS MYSEA, IPv4, IPv6, MLS, IP next generation, Multilevel security, Network Address Translation, NAT			**15. NUMBER OF PAGES** 85
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A DESIGN COMPARISON BETWEEN IPV4 AND IPV6 IN THE CONTEXT OF MYSEA, AND IMPLEMENTATION OF AN IPV6 MYSEA PROTOTYPE

Matthew R. O'Neal
Ensign, United States Navy
B.S., United States Naval Academy, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2003

Author: Matthew R. O'Neal

Approved by: Dr. Cynthia E. Irvine
Thesis Advisor

Thuy D. Nguyen
Second Reader

Dr. Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Internet Protocol version six (IPv6) is only sparsely implemented in the world today. However, as it gains popularity, it will grow into a vital part of the Internet and communications technology in general. Many large organizations, including the Department of Defense, are considering deployment of IPv6. Experiments will ensure that IPv6 will work with both existing and planned applications. One area where its success is essential is that of systems designed to support multilevel security.

This thesis focuses on the design and implementation issues that accompany a migration of the Monterey Security Enhanced Architecture (MYSEA) from Internet Protocol version four (IPv4) to IPv6. The research for this thesis consists of two major parts: a functional comparison between the IPv4 and IPv6 designs, and a prototype implementation of MYSEA in an IPv6 environment.

The current MYSEA prototype relies on a subset of Network Address Translation (NAT) functionality to support the network's operation; and, since IPv6 has no native support for it, a NAT mechanism was created in IPv6.

This thesis provides a preliminary examination of IPv6 in MYSEA, which is a necessary step in determining whether the new protocol will assist with or detract from the enforcement of MYSEA policies.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

MYSEA	Monterey Security Enhanced Architecture
MLS	Multilevel Security
TPE	Trusted Path Extension
SAR	Secure Attention Request
IP	Internet Protocol
IPv4	Internet Protocol version four
IPv6	Internet Protocol version six
MTU	Maximum Transmission Unit
NAT	Network Address Translation
PAT	Port Address Translation
IPsec	Internet Protocol security
SPD	Security Policy Database
SA	Security Association
ISAKMP	Internet Security Association Key Management Protocol
IKE	Internet Key Exchange
QoS	Quality of Service
QoSS	Quality of Security Service
QoSM	Quality of Service Manager
MAC	Mandatory Access Controls
DAC	Discretionary Access Controls
TCB	Trusted Computing Base
NAT-PT	Network Address Translation – Protocol Translation
TRT	Transport Relay Translator
6to4	Six-to-Four
COTS	Commercial Off-the-shelf
DoD	Department of Defense
FTP	File Transfer Protocol
I/O	Input / Output
ISP	Internet Service Provider
LAN	Local Area Network
NIC	Network Interface Card
OS	Operating System
PC	Personal Computer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UID	User Identity

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Thank you to my thesis advisor, Dr. Irvine, for so much inspiration and strict guidance during my work on this project.

Many thanks to David Shifflett, without whom I would probably still be hacking around with Unix commands and writing C code.

Thanks also to Thuy Nguyen, my second reader, for being so demanding when it came to the precision of my writing. You were like my conscience, helping me keep everything straight.

I very much appreciate my wife, Aubrena, for remaining supportive of me and inspiring me to keep trying to get smarter.

Of course, I can't forget Mom. Even from so far away and with her own things going on, Mom finds time, energy, and ways to remind that she loves me and supports me. Thanks, Lady.

Finally, and most importantly, I am thankful to the good Lord for blessing me with great opportunities and the abilities to excel at them.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

In the Internet Protocol version six (IPv6), also known as the next generation Internet Protocol, lies the future of communications for networked computers and possibly the future of all telecommunications. Designed to augment and eventually replace the aging Internet Protocol version four (IPv4), the current standard, IPv6 stands in a position to replace the more than two-decade-old Internet Protocol (IP). The design of IPv6 likely contains improvements over the drawbacks of IPv4, some of which are causing concern among the community of Internet designers and engineers. Two examples of these trouble areas are the shrinking of the pool of available IP addresses, and the growth in size of routing tables stored on Internet routers. With time, the IP address space is becoming more and more stretched because of the unanticipated growth of the Internet. The growth of routing tables is attributable to inefficiencies of the initial IP addressing hierarchy. The web address cited in [PROBLEM] provides a synopsis of the history of the Internet's addressing troubles, and RFC 1752 [REC_IPng] provides a history of the birth of IPv6, including why it was developed. Additionally, new features in IPv6 may help to augment security and/or help IP to provide improved services.

While IPv6 differs from IPv4, it is designed to perform the same basic functions as the original Internet Protocol. With this fact in mind, it is natural to hypothesize that the design of IPv6 improves on the original IP design while not adversely affecting the services it provides. The vastly larger address space and the native support for Internet Protocol Security (IPSEC) are two positive changes IPv6.

A. PURPOSE OF STUDY

There exist multiple reasons for performing this study. First of all, the Department of Defense (DoD) has committed itself to full deployment of Internet Protocol version six (IPv6) by the 2008 fiscal year [MEMO]. Secondly, the Internet is in the beginning stages of a transition to IPv6. Finally, new features in IPv6 have the potential to improve IP services in various applications. A clear determination of this potential is necessary before transitioning systems to IPv6.

The Monterey Security Enhanced Architecture (MYSEA) is a multilevel secure local area network (MLS LAN) that is designed to manage data at various levels of classification, and to allow untrusted commercial-off-the-shelf (COTS) client machines to securely access that data. This research specifically focuses on the design considerations of running MYSEA on an IPv6 network vice an IPv4 network. From a design perspective, it explores the areas in which IPv6 can assist in MYSEA's ability to enforce network policy.

In anticipation of making a transition to IPv6, it is necessary to analyze the costs and benefits of running MYSEA on an IPv6 network. Building MYSEA with native IPv6 functionality may even support and benefit the architecture more than IPv4. For a system like MYSEA to successfully complete a transition from IPv4 to IPv6, its designers and implementers must prepare early and understand any modifications this transition will demand. The research documented in this paper will provide the foundation of the work to build MYSEA in an IPv6 environment.

This work includes a review and comparison of the IPv4 and IPv6 designs. In addition, an IPv6 MYSEA prototype has also been developed. The MYSEA design requires functionality that is provided by network address translation (NAT) in IPv6; however, there currently are no – and there likely never will be any – NAT mechanisms defined or implemented for IPv6. This situation required either finding a replacement mechanism for NAT or implementing NAT in IPv6. The implementation chapter, Chapter IV, contains the details of this work.

B. OVERVIEW OF CHAPTERS

This section contains an overview of the remaining chapters of this paper.

1. Chapter II, "The Monterey Security Enhanced Architecture (MYSEA)"

Beginning with brief description and the reasoning for its conception, Chapter II provides a fairly detailed description of the Monterey Security Enhanced Architecture. The chapter outlines the design of each component, stating the status of each in the current prototype, and it ends with a brief discussion of the intention for MYSEA to support the provision of Quality of Security Service (QoSS).

2. Chapter III, "The Internet Protocol"

Chapter III summarizes the design specifics of Internet Protocol versions four and six. It also briefly discusses Internet Protocol security (IPsec) and its disposition in MYSEA with both IPv4 and IPv6. Next, it contains a discussion of some mechanisms currently being used and developed for the transition of the Internet to IPv6; and lastly, it presents the results of the comparison between the designs of the two protocols.

3. Chapter IV, "Implementation of IPv6 MYSEA Prototype"

This chapter describes the work involved in implementing the IPv6 MYSEA network for this thesis.

4. Chapter V, "Conclusions and Future Work"

Chapter V contains the overall conclusions from the thesis work and it summarizes the conclusions from the analysis of the IPv4 and IPv6 designs. It concludes with a discussion of future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. THE MONTEREY SECURITY ENHANCED ARCHITECTURE (MYSEA)

The Monterey Security Enhanced Architecture (MYSEA) manages information at several classification levels, and it is built upon trusted security services and integrated operating system mechanisms [MYSEA ARCH]. The system as a whole is intended to act as a high-assurance distributed operating environment that enforces a multilevel security policy. It is designed to interoperate with commercial computer components and to make use of pre-existing productivity tools. Such a design allows the current DoD investment in commercial personal computers (PCs) to be integrated into an environment where more trusted elements enforce critical security policies [MYSEA PROJ]. The system configuration supports traditional multilevel secure (MLS) networks, coalition networks, joint ventures between various departments of the U.S. government, and enforcement of mandatory policies.

A. THE REASON FOR MYSEA

The architectural design of MYSEA responds to the problem of having to maintain separate networks in order to handle data of differing security levels or coalition classifications. Take, for example, the coalition problem. A coalition consists of a diverse group of entities with a common goal. Each of these entities must have the ability to exclusively share information with its own members, and each will invariably share some information with the rest of the coalition and the coalition commander. There exist two approaches for providing the desired services.

1. Build a physically separated network for each entity that allows exclusive communications between its members, and build another stand-alone network to which all entities have access. This second network allows information sharing across entities, but requires a "sneaker-net" mechanism in order to move information to and from the shared network.

2. Construct a single MLS network, relying on trusted components to enforce network policy, and use security labels to place each entity's private data into a separate security level. Other labels permit information sharing across entities.

While effective in separating data, the first approach requires a great deal more hardware and physical space than the second. Every individual would require a different computer workstation for every network to which he has access. In addition, the number of networks and workstations increases as the number of coalition members increases. The second approach provides precisely the same functionality as the first, but with fewer physical resources and far fewer infrastructure requirements. In MYSEA, every data object, a file for instance, carries a security label that corresponds to a predefined security level. A user with authorization to access a particular data object must be logged in at a security level that is equal to or greater than the security level of that object's label. In true MLS, a user can read data at his current security level as well as all levels below that. The term "logged in" refers to using some means of authenticating oneself to the system, e.g., password, biometrics, retinal scan, etc. Still, a potentially large problem for both approaches is a lack of assurance that separation and integrity of network data is preserved. MYSEA overcomes that shortfall through the use of high assurance policy enforcing components.

The realization of the concepts that drive MYSEA's design and implementation will mitigate the current situation of government and DoD systems not providing adequate security and assurance for critical data. As stated in [MYSEA ARCH], "industrial systems run the risk of economic espionage, while the lack of policy-enabled Joint Command and Control Systems constrains military operations." The following two reasons largely contribute to the lack of adequate security on proprietary and open-source computer systems:

1. The use of discretionary access controls (DAC, described below) have become widespread.

2. Flaws in system design and / or implementation create unforeseen security holes that require patches after the deployment of a system.

A system that depends on DAC allows users to dictate access control policy (i.e., set file permissions) "on the fly;" and, as a result, it is highly susceptible to Trojan Horse attacks. In these attacks, a malicious program performs harmful actions under the identity of a compromised user, so it is able to manipulate the permissions of all objects

to which that user has access. Mandatory Access Controls (MAC) present an alternative security enforcement mechanism for a system. In contrast to a DAC system, a system enforcing a MAC policy prohibits malicious software from performing actions that violate the system's security policy (e.g., modifying file permissions and giving all user identities (UIDs) read access to a file to which a limited number of UIDs should have access). A robust security policy will disallow applications acting on behalf of a user from performing potentially malicious actions such as the permissions of a restricted file. The word restricted refers to objects to which a limited number of entities should have access.

Systems released without sufficient security engineering, and formal code verification in some cases, frequently require subsequent system patches to correct unwanted behavior. In particular, these systems risk subversion (the insertion of a backdoor) by a malicious insider. The existence of "Easter Eggs" in popular programs serves as an illustration of the potential for malicious developers to insert backdoors into programs and operating systems (see [SUBVERSION]).

MYSEA's design makes security a non-functional requirement along with the functional requirements. Security remains a primary focus throughout the system development process. In MYSEA, the machine that stores and serves data enforces a multilevel policy. The policy is global and persistent; therefore, it affects all users of the system and remains in place for all network operations.

B. DESIGN AND COMPONENTS

The MYSEA architecture consists of the following major parts, which are illustrated in Figure 1:

- A high assurance server enforcing a MLS policy.
- Commercial off-the-shelf (COTS) PC workstations.
- A Trusted Path Extension (TPE) device between each workstation and the rest of the network. [MYSEA ARCH]

The TPE creates a trusted path between the workstation and the trusted computing base (TCB) at the server. A trusted path ensures that only the authentication device receives the user's authentication data. Moreover, the TPE provides a guaranteed

authentic connection for the client workstation and server, and it guarantees secure communications between the two. Figure 1 depicts a MYSEA network set up in the way that a coalition network might appear. The shadings illustrate the fact that each member only has access to his own organization's data. The multi-shaded workstation represents a coalition command workstation from which authorized users have access to all levels of data. The pattern in the MLS server and the TPEs indicates that they are trusted components.

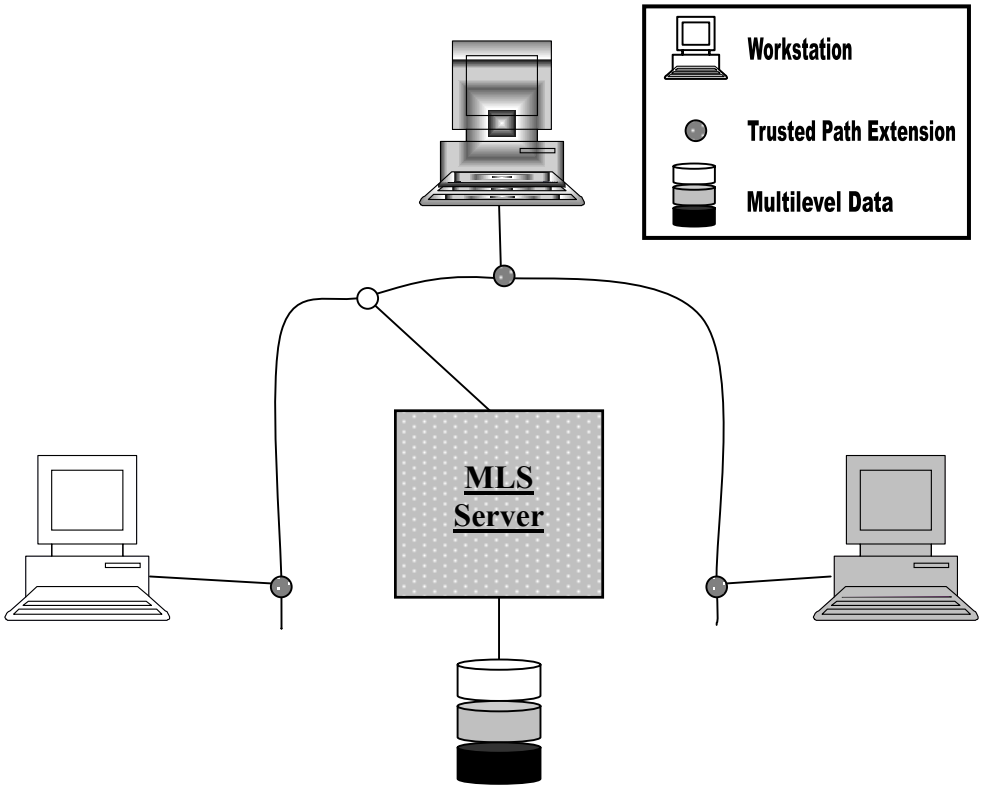

Figure 1. MYSEA design

While this architecture consists of three major parts, the design requires the server to view the TPE and workstation as a single entity. In essence, the TPE becomes a necessary component of the workstation. It cuts off workstation communications until a user authenticates himself to the server through the TPE, thereby opening the communications channel and allowing the person to access the server through the workstation

Note that this discussion of MYSEA serves only as a cursory view of the purpose, design, and implementation of the architecture. Refer to [MYSEA PROJ] and [MYSEA ARCH] for more in-depth discussions.

1. The MYSEA Server

In the current prototype, the MYSEA server runs on an Intel x86 PC whose operating system (OS) is a modified version of OpenBSD. The modifications enable the OS to add security labels to the data it stores. The OS enforces the security policy while supporting the use of untrusted application servers that are compatible with existing COTS clients. For instance, a derivation of the popular Apache web server may run on top of the OS and serve web pages to users who are logged in to the workstations; however, in spite of any security flaws existing in the server application, given that OS enforces true MLS, it cannot break the OS-enforced security policy. The data that the server's OS maintains is labeled to separate it into security domains such as classified and unclassified. The ultimate implementation of the server will exist on a high assurance commercial server.

2. The MYSEA Workstation

The ultimate vision of the MYSEA workstation is a diskless machine containing the following elements: a processor, a user interface, and I/O services that support network communications and data transfer [MYSEA ARCH]. The existence of the TPE allows the PC to be an untrusted device. Since the PC is not trusted, the specific OS of the workstation can be anything with which a user is comfortable. The architecture intends that the PCs have the ability to support application protocol clients, such as a web browser, so that users can access data available through application protocol servers running on the MYSEA servers. An important aspect of the workstations is that they are intended to be diskless machines with enough random access memory (RAM) to support a user's client applications. A diskless workstation helps to prevent storage of sensitive information at the workstation. The RAM and other storage media on the workstation, if any, will be purged when a user changes session levels or logs out.

3. The Trusted Path Extension (TPE)

As stated above, the TPE exists logically as a part of the workstation; however, from a design perspective, the TPE is obviously a separate entity. The workstation is an untrusted entity, so the TPE extends the server's trust to the user through the trusted path that it creates. It physically resides between the workstation and the server, acting as the only conduit through which the workstation can reach the rest of the network.

The TPEs Functionality includes controlling the workstation, securely handling I/O from the user, and maintaining a trusted path connection with the server. Through the TPE's I/O interface, the user authenticates himself to the server by password, biometrics, or some other form of identification and authentication. After successful authentication, he contacts the server through the TPE and requests a session level at which to operate. When the user receives permission to operate at the requested session level, the workstation becomes accessible and the user conducts business at the selected security level. The current MYSEA prototype implements the TPE on a handheld PC equipped with two network interface cards (NICs). This implementation requires the workstation and the server to lie on two distinctly separate networks. While this separation would likely be the natural case in the deployable MYSEA implementation, the use of two NICs in the TPE causes it to be a requirement in this prototype. The TPE is truly a bridge between the server and workstation as it acts as a pseudo-router and routes packets between the server and client networks. It overcomes physical separation from the workstation by gaining a logical oneness (from the server's point of view) with it through the use of NAT. Because of NAT, the server only explicitly communicates with the TPE, knowing nothing of the existence of the workstation behind it. When a user is logged in, the TPE forwards relevant traffic between the workstation and the server (See the NAT discussion below for more details on network address translation).

Although the trusted path extension currently resides on a handheld PC that runs the Linux OS, it will ultimately not run on a general-purpose operating system. The TPE's required functionality does not warrant use of the complex tools and services that a full operating system generally provides. Therefore, the TPE will ultimately be an

embedded system that essentially functions as an extension of the server. It will have obtained a high assurance, EAL7 evaluation with EAL6 applications.

C. QUALITY OF SECURITY SERVICE (QOSS)

The MYSEA architecture, specifically the server, is intended to support integration with an external quality of service manager (QoSM). The server provides a security-aware interface to this external resource manager called the MYSEA QoSS manager.

The notion of QoSS has to do with managing security-related functionality, such as amount or level of encryption, as a QoS parameter. Implementation of this idea means defining variable levels of security services for a network. Similar to response time and image fidelity, variable levels of security services and requirements can be presented to users or network processes in the form of acceptable ranges [QOSS]. With these security-related variables, the QoSM has a greater ability to meet overall user and network demands.

THIS PAGE INTENTIONALLY LEFT BLANK

III. THE INTERNET PROTOCOL

The Internet Protocol has served as the communications standard of the Internet since its birth more than two decades ago. This chapter contains summaries of the designs of the current Internet Protocol, IPv4, and the future Internet Protocol, IPv6. A discussion of IPsec, a protocol suite that addresses security services provided at the network layer of communications, follows those design summaries. Following the IPsec section is the comparison of the IPv4 and IPv6 protocols, and the final section of this chapter contains an introduction to the proposed IPv4-to-IPv6 transition mechanisms for the Internet.

A. INTERNET PROTOCOL VERSION FOUR (IPV4)

This section presents an overview of IPv4 beginning with the purpose for its development. It assumes the reader has some familiarity with the TCP/IP stack.

1. Motivation

The Internet Protocol was developed to provide a mechanism for the transmission of data between hosts on interconnected, packet-switched networks [IP]. Before IP, it was impossible for hosts on different types of networks to communicate. For example, a host on a Token Ring network could not transmit data to a host on an Ethernet. This inability to communicate exists because of incompatibilities such as differing transmission speeds, signaling methods, and synchronization techniques. IP provides a universal means of communications for host computers that reside on different types of networks, as well as for hosts that are separated by a large geographic margin. The protocol's assumptions are as follows: both hosts understand IP; each host understands the communications protocol that its respective local network uses; each host has the ability to communicate with a router, possibly part of a group of routers, with the ability to forward data from one host network to the other. Since the two hosts cannot necessarily communicate through their native protocols, they must use IP to carry their data, and they must depend on the router(s) to deliver that data to the correct destination.

The Internet Protocol design has an intentionally limited scope to perform two primary functions: addressing and fragmentation [IP]. Every host must have an IP

address, and all IP addresses must conform to the IP addressing hierarchy defined in [IP] and discussed below. Fragmentation refers to the breaking of large datagrams into fragments so that each one is no larger than a network's maximum transmission unit (MTU). While IP has no error control abilities, it depends on ICMP (Internet Control Message Protocol) for error reporting. The IP implementation treats ICMP like an upper level protocol, but it is an integral part of IP, and all hosts and routers must implement it [ICMP].

2. Header Structure

Each IP datagram begins with the IP header, and the payload data follows immediately thereafter. This header contains identifying data for the datagram as well as useful information for processing that data. Figure 2 contains a simple illustration of the IPv4 header.

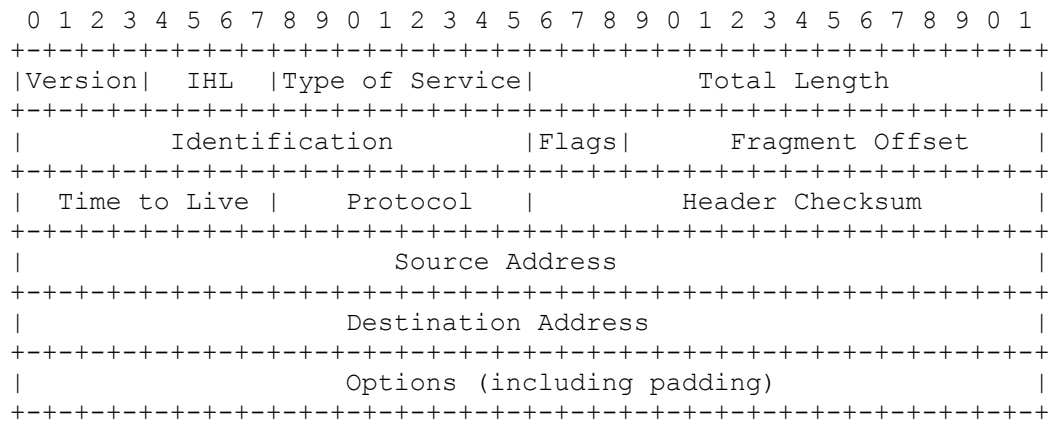

Figure 2. IPv4 Header [IP]

The dashes between the plus signs represent one bit, and the numbers above the figure indicate a bit count (modulus 10). Each row is thirty-two bits long. The words in the fields convey what the bits in that field represent. A brief description of each field follows:

- Version: The Internet Protocol version
- IHL: Internet header length. This field contains the length of the header in thirty-two bit words.
- Type of Service: Bits in this field specify abstract quality of service parameters.
- Total Length: The total length of the datagram in bytes.
- Identification: A value that uniquely identifies each datagram sent by a host.
- Flags: Control flags. Specifically, a Don't Fragment bit, and a More Fragments bit.

- Fragment offset: This field indicates the position of a fragment relative to other fragments with the same identification value.
- Time to live (TTL): Specifies the number of routers the datagram can pass through. Each time a router processes a datagram, it decrements this value. When the TTL reaches zero, the next router discards the datagram.
- Protocol: This field specifies the next level protocol in the payload following the header – TCP, for instance.
- Header checksum: An error detection checksum for the IP header.
- Source address: The originating IP address of the datagram.
- Destination address: The final destination IP address of the datagram.
- Options: The sender may specify various options within this field. It has a variable length and may require padding to ensure that it ends on a thirty-two bit boundary.

See [IP] for a more in-depth discussion of the IP header and the fields within it.

a. Options

The options field of the IPv4 header is of variable length. This variability is because a variable number of options may exist, and nearly every option has a variable non-static size. Following is the list of available options contained in [IP]– not including options that perform no operations – along with a brief description:

- Security – Contains security and compartmentation information.
- Loose Source Routing – Specifies a route for the IP datagram to follow, but allowing for non-specified nodes to process the datagram.
- Strict Source Routing – Specifies a strict route for the datagram to follow, intending that no other nodes process the datagram.
- Record Route – Records the IP address of each node that processes the datagram.
- Stream ID – Designed to carry a sixteen bit SATNET stream identifier through networks that do not support the stream concept.
- Internet Timestamp – Each forwarding node inserts a timestamp into this field.

3. Security

Speaking only in terms of authentication and data encryption, the original design of IP totally lacks security mechanisms. The security option, described above, only exists as a means of compartmenting the data carried in the datagram. Furthermore, the "security" that it provides is effective only so long as the receivers of IP traffic obey the standard. Bear in mind, however, that just as IP depends on upper level protocols for

functions like error control, those higher-level protocols may also provide the necessary functionality (such as encryption) for securing the payload.

Applications can provide the means of ensuring data confidentiality, integrity, and authenticity. Part of the problem with depending on applications is that there is no standard, and the methods of securing data vary from application to application. Furthermore, applications sometimes fail in providing adequate security, and the results of such failure can be disastrous. Besides, in MYSEA, as is the case in nearly all trusted computing frameworks, the applications are untrusted. This means that all applications, malicious or not, are treated as though they might be malicious. This treatment results from the simple fact that there is no guarantee that the application performs all of its declared operations and nothing more. Designers of a TCB would actually prefer that applications not perform such actions as encryption because the TCB must be able to enforce the security policy on inbound and outbound data.

4. Addressing Architecture

In text representation, one may represent an IP address in one of two forms -- bitwise notation and dotted decimal notation. Dotted decimal is the more widely used and more recognizable notation. Each IP address may be logically separated into two parts. The first group of bits represents the network address, and the remaining bits denote the IP address of a host on that network. The network mask is the set of bits that distinguishes which portion of the address identifies the network and which portion identifies the node [CISCO]. Figure 3 shows a Class A address along with its network mask in both dotted decimal and bitwise formats.

```
8.20.15.1 = 00001000.00010100.00001111.00000001
255.0.0.0 = 11111111.00000000.00000000.00000000
-----------------------------------------------
              NET ID | <- HOST ID ->
```

Figure 3. IP address and Mask example

The addressing architecture separates the address space into five classes represented by the letters A, B, C, D, and E. Class D is reserved for multicast traffic, and class E is reserved for future use. In a class A address, the first eight bits represent the

network, and the remaining twenty-four bits represent a host's address. The first sixteen bits represent the network in a class B addresses, and the first twenty-four bits represent the network in a class C address. Figure 4 presents a pictorial view of the address classes. The ones and zeros on the far left of each address block show the leading bits of the IP address, which are a way of distinguishing the address class. The numbers to the right of each class give the range of network addresses, in dotted decimal form, that the class covers.

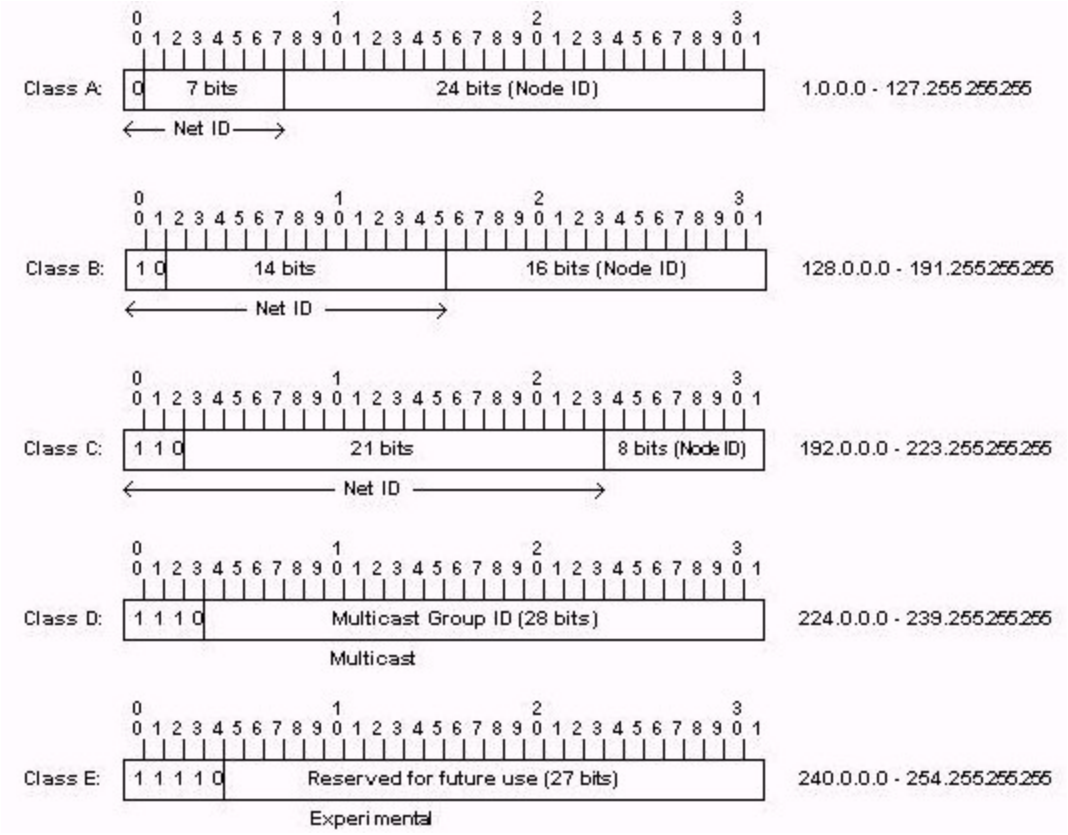

Figure 4. IP Addressing scheme [CISCO]

Note that if a node sends a datagram to a network's address, it is considered a broadcast, and the network's border router (the network's gateway to the internet) will forward the datagram to every host on the network. For instance, consider a class A network with the network ID of 10.0.0.0 containing two hosts with the addresses 10.1.1.1 and 10.2.2.2. If the border router receives a datagram addressed to 10.0.0.0, it will forward the datagram to every existing host inside the network. In this case both 10.1.1.1 and 10.2.2.2 will receive the datagram.

In addition to the classful hierarchy shown in Figure 4, IP contains a classless element that allows routing decisions to be made based on the address mask instead of the class of the IP address. The reason for the introduction of classless IP is that traditional subnetting wastes IP addresses [CISCO]; and due to the rapid growth of the internet, IP addresses are becoming scarcer. The two methods used to apply this classless structure are Variable Length Subnet Masking (VLSM) and Classless Inter-Domain Routing (CIDR). Both may be used in the classful structure, but they enable more efficient use of the IP address space. Ever since the explosion of the Internet around the world in the 1990s, the address space has become an area of growing concern among Internet designers and developers. VLSM and CIDR help to make better use of the entire address space, increasing the potential for using every possible IP address. For in-depth discussions of IP addressing, VLSM, and CIDR, see references [3COM] and [CISCO]. The next section deals with another approach to addressing the concerns with the IP address space.

5. Network Address Translation (NAT)

The development and deployment of NAT has come with many different benefits, and even some drawbacks. As explained in RFC 2663, "The term 'Network Address Translator' means different things in different contexts" [NAT_TERM]. The intent of this section is not to describe the many varieties, uses, advantages, and disadvantages of NAT; but merely to introduce the concept that it implements.

a. NAT Defined

Network Address Translation is a mechanism that allows nodes bearing private (unregistered) IP addresses to communicate in the global Internet by replacing the private addresses with public (globally unique) ones. The following paragraph illustrates the key ideas of NAT.

In a private network (using private IP addresses) that runs NAT, the border routers implement the NAT functionality. Normally, a border router will not forward any datagrams from an intranet into the Internet because they contain a private IP address as the source; however, a NAT router will simply swap the private address for a predetermined public address that conforms to the standard – either its own global

address, or one from a pool of allocated valid addresses. After forwarding the modified datagram, the router maintains the address mapping so that it can map the reply packets to the substituted address. That is the basic function of NAT.

Figure 5 and the example below it use the MYSEA architecture to illustrate how routers perform NAT.

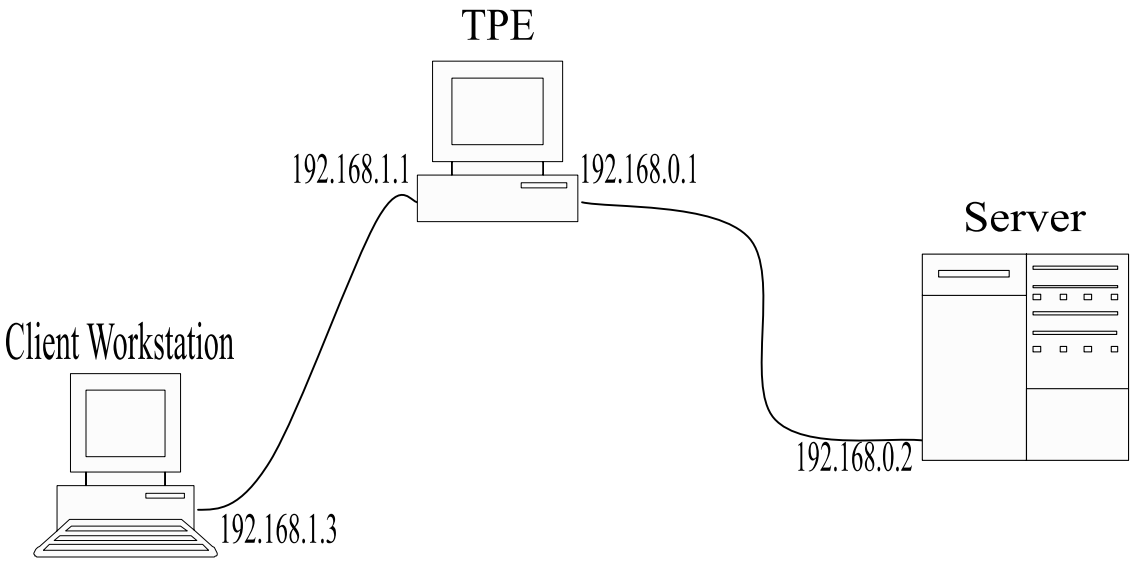

Figure 5. NAT example

Each IP address in Figure 5 lies next to the interface to which it is bound, and they all have a 255.255.255.0 network mask. In this example, all of the IP addresses reside in the private address space. The actual address values are arbitrary, and are included in this example as references. Recall that the TPE performs the same functions as a border router. When the client sends a datagram to the server, the TPE modifies the source address of that datagram before forwarding it. In the above MYSEA situation, the new source address of the client's datagram that the TPE forwards to the server will be 192.168.0.1. In addition to replacing the source address, the TPE creates and maintains a mapping between the replacement address and the information that uniquely identifies the session (e.g. IP addresses and port numbers). From then on until the session ends or times out, the TPE imposes the address mapping on all traffic that matches a session identifier. When the server responds to the client's datagram, it responds to 192.168.0.1.

Upon reception of that datagram, the TPE finds that it matches the existing session, so it replaces the datagram's destination address with 192.168.1.3 and forwards it to the client.

b. NAT & MYSEA

While an ongoing debate rages over the utility and drawbacks of using NAT, the mechanism has actually proven very useful for MYSEA. A useful feature of NAT, more accurately referred to as a side effect, is the fact that it allows a border router to hide internal IP addresses from the global Internet. NAT must perform this action in order to precisely perform its service of disallowing the propagation of privately sourced traffic into the internet; and MYSEA takes advantage of this side effect to create a logical union between the TPE and the client. With NAT, the server only "knows" one IP address that logically refers to the client, but is physically assigned to the TPE, allowing the TPE to remain in the loop and ensure that MYSEA policies are maintained.

B. INTERNET PROTOCOL VERSION SIX (IPV6)

IPv6 represents the next step in the evolution of a robust, flexible communications protocol that is intended to accommodate the communications and information sharing needs of the world. This section contains a summary of the IPv6 specification, [IP6]. The information herein focuses on IPv6 as it applies to MYSEA and with regard to IPv4. By no means does this section contain a comprehensive description of the protocol. For more details on IPv6 see [IP6].

1. General Changes to the IP Design

Note that the designers of IPv6 do not make any fundamental changes to the basic concept and functionality that the Internet Protocol intends to provide. IPv6 retains the same scope as IPv4, but the new design attempts to improve on the original design by making it simpler, yet more flexible, and no harder to implement. The following list, presented in [IP6], summarizes the intended changes from IPv4 to IPv6:

- Expanded addressing capability: The address size has increased from 32 to 128 bits. The new design also contains some changes to addressing schemes and address assignment that are beyond the scope of this discussion.
- Simplified header format: Discussed in the following section.

- Better support for extensions and options: The specification changes the encoding of IP header options, thereby increasing efficiency and flexibility, and easing the introduction of new options in the future.

- A flow labeling capability: A capability for labeling packets that belong to particular traffic flows for which a sender requests special handling.

- Privacy and authentication capabilities: IPv6 provides explicit extensions to support authentication, integrity, and confidentiality.

This list contains the intended changes from IPv4 to IPv6. Other significant changes in IPv6 include the assumption that every link in the Internet has an MTU of at least 1280 bytes. Also, only the originating node of a packet may perform fragmentation. The following sections will elaborate on the intended changes while providing an overview of IPv6.

2. IPv6 Headers

As previously stated, the format of the IPv6 header is a simplified version of the IPv4 header. Figure 6 illustrates the IPv6 header structure.

As with the IPv4 header depiction, the numbers above the illustration represent a bit count, beginning with the number zero. The minimum size of an IPv6 header is 40 bytes, twice the size of the IPv4 header. The large size of the addresses almost necessitates simplifying and making the rest of the header smaller for the sake of conserving bandwidth.

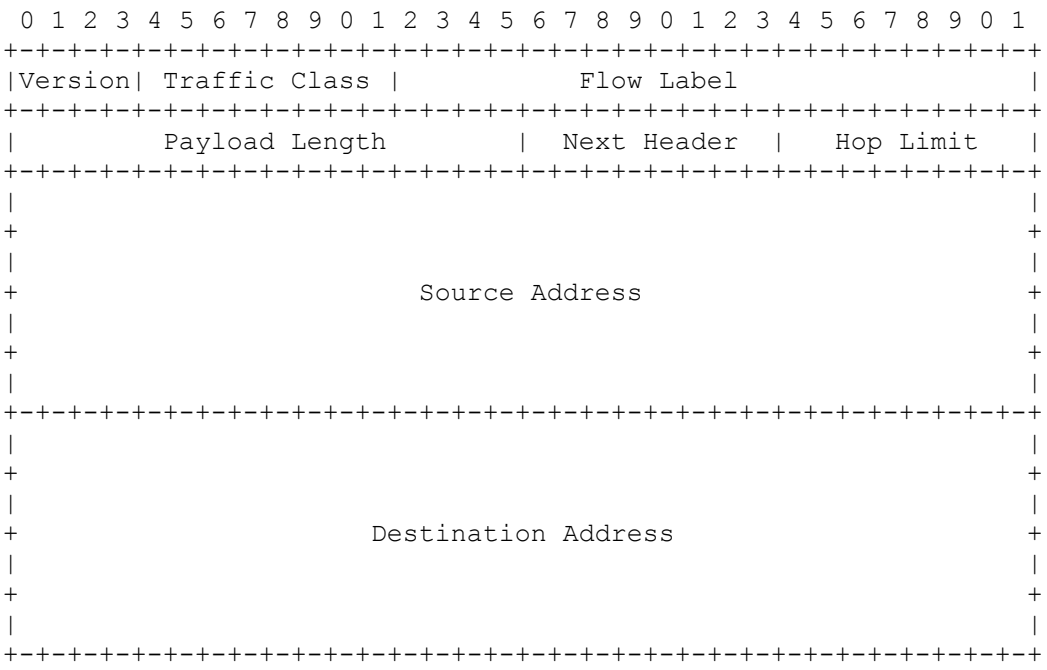

Figure 6. IPv6 Header [IP6]

The following list contains a brief description of each field in the header:

- Version: Current IP version
- Traffic Class: For use in distinguishing between classes or priorities of packets. This field is equivalent to the IPv4 TOS field.
- Flow Label: This field contains a label assigned to sequences of packets that require special handling by routers, such as a QoS specification.
- Payload Length: This field specifies the length, in bytes, of the payload, that is everything following the IPv6 header.
- Next Header: Specifies the type of header following the IPv6 header.
- Hop Limit:Performs the same function as the IPv4 TTL field. Each forwarding node decrements this value by one.
- Source Address: The 128-bit address of the originator of the packet.
- Destination Address: The address of the intended recipient of the packet.

IPv6 uses extension headers to encode optional information at the network layer, thereby adding to the modularity of the IP design. These headers lie between the IPv6 header and the next layer protocol header in an IPv6 packet. Figure 7 illustrates the use of extension headers.

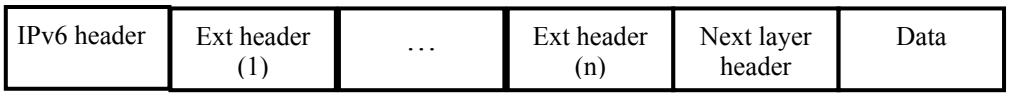

Figure 7. IPv6 Extension header example

This capability, in part, replaces the functionality of the variable-sized options field in the IPv4 header. Since all fields in the IPv6 header have a fixed-length, the IPv6 header has a truly static size. An IPv6 packet can contain zero or more extension headers. Following is the list of extension headers specified in [IP6]:

- Hop-by-Hop Options
- Routing
- Fragment
- Destination Options
- Authentication
- Encapsulating Security Payload (ESP)

Nodes that forward packets do not examine any of these headers, with the exception of the Hop-by-Hop Options header. Every node in a packet's path from source to destination always examines this header. The specification adds more structure to the use of the extension headers by setting a specific order in which to include them (see [IP6] for that order). The Routing header provides functionality similar to the Loose Source Routing, Strict Source Routing, and Record Route options in IPv4. The IPv4 section of this paper contains short descriptions of those routing options. Source nodes include a Fragment header with each fragment of a transmitted packet. The Destination Options header carries optional information that only the ultimate receiver of a packet inspects. Options following this header have a variable length. The specification currently defines two options dealing with padding. It also provides some initial structure – required values of high-order bits for unrecognized options – for option definitions, and it contains guidance for introducing new options. Finally, the Authentication and ESP headers provide authentication and encryption respectively. These two headers relate directly to IPsec, and they are discussed in the IPsec section.

3. Addressing Architecture

RFC 2373 [IP6 ADDR] is the primary resource for IPv6 addressing, and the majority of IPv6 addressing information resides in that document. The model for addressing in IPv6 closely resembles that of IPv4, except that it natively employs the concept of CIDR. The 128-bit address, the native use of CIDR, and a new IPv6 addressing model stand out the most.

a. Basic Differences from IPv4 Addressing

As stated above, IPv6 uses a classless addressing structure. While the hierarchy is classless, IPv6 still has various ranges of reserved IP addresses. The specification also defines the following three address types for IPv6: unicast, anycast, and multicast. A unicast address simply identifies a single interface and functions as a normal IP address, the same as IPv4 addresses. An anycast address identifies a set of interfaces (usually on different nodes), and the "nearest" one, according to the routing protocol, receives the so addressed packet. IPv4 has no inherent provision for anycast addresses. A multicast address also identifies multiple interfaces that normally lie on different machines. A packet destined for this type of address is accepted by all interfaces that share the address. The IPv6 multicast address overrides IPv4's broadcast capability, so there are no broadcast addresses in IPv6. Note that IPv4 does have a specified multicast capability which was developed after the initial IP addressing specification.

Finally, the format for representing an IPv6 address in text differs from the IPv4 format. While it is possible to represent an IPv6 address in bitwise or dotted decimal notation, it would be much harder for a human reader to interpret since an IPv6 address is eight times larger than an IPv4 address. Instead, the standard separates an IPv6 address into eight pieces, each one represented by a sixteen bit hexadecimal value. A colon separates each value. A common shorthand method for representing multiple sequential zeros is presented in Figure 8. Alternatively, one may specify the first ninety-six bits using hexadecimal values and then use the well-known IPv4 bitwise notation to represent the final thirty-two bits. This format is useful for representing IPv6 addresses that map directly into IPv4 addresses. The section on transition tools discusses this type

of IPv6 address. There are other minor intricacies involved with representing these addresses, but this is the basic method. More information is contained in [IP6 ADDR].

```
fec0:0:0:0:0:0:0:5

is the same as

fec0::5
```

Figure 8. IPv6 Address Examples

b. General Addressing Information

Similar to IPv4 private addresses, IPv6 defines two types of "local-use" [IP6 ADDR] addresses that are designated by the first ten bits in the IPv6 address. In this context, local-use refers to addresses in packets that routers must not forward. A Link-Local address serves as an address for a host on a given link in a network, and packets containing these addresses must not be forwarded beyond that link. Site-Local addresses may exist beyond a single link, but must not be forwarded beyond the site from which they originate. Additionally, [AGGR] defines an IP address aggregation scheme for IPv6 global unicast addresses. The intent of defining this scheme is to streamline Internet routing tables and reduce complexity in anticipation of a larger, more complex Internet in the future. Refer to [IP6 ADDR] and [AGGR] for more detailed information on IPv6 addressing and global address aggregation, respectively.

4. Security

As stated in the list of changes in IPv6, the protocol was designed with authentication and privacy capabilities. The Authentication and ESP extension headers provide these capabilities through the functions they perform, and these two headers are actually a part of the separately defined IP security architecture. This architecture is laid out and discussed in [IPSEC ARCH], and is briefly discussed in Section C. Based on the release dates of the RFCs, the security architecture existed before the IPv6 specification was finalized. Therefore, since IPv6 incorporates IPsec into its design, it is accurate to state that IPv6 provides native support for authentication and confidentiality (encryption) of data. Section C introduces the IPsec architecture and describes the functions that the Authentication and ESP headers provide.

C. INTERNET PROTOCOL SECURITY (IPsec)

The Internet Protocol security architecture, better known as IPsec, has the capability to provide two essential functions to MYSEA. IPsec's encryption capabilities protect data flowing between the Trusted Path Extension and the server, and its authentication capabilities provide two-way authentication between those two nodes. Additionally, IPsec is the mechanism chosen to support the Quality of Security Service requirements discussed in Chapter I.

The remainder of this section presents an overview of the IPsec design, framework, and its implementation in MYSEA. IPsec and all of its supporting concepts and operations are defined in multiple documents with a lot of intertwining information. This section attempts to capture the overall essence of IPsec without delving too deeply into the great amount of information defining it. The information provided here on IPsec is drawn from [IPsec, ISAKMP, and IKE]. Refer to [DOCMAP] for a listing of the documents pertaining to IPsec and a description of their interrelationships.

1. Design

IPsec is intended to provide a common set of security services for nodes on the Internet. These services are listed in the following sub-section. The major advantage of providing security services at the IP layer is that the services are available for IP traffic and all higher layer protocols [IPsec]. Since the Internet Protocol is standardized throughout the Internet, the IPsec services are universally available.

a. Goal

The design goal of IPsec aims to provide "interoperable, high quality, cryptographically-based security for IPv4 and IPv6" [IPsec]. IPsec provides the following services as described in [IPsec]:

- Access control
- Connectionless integrity
- Authentication
- Replay protection
- Data confidentiality
- Limited traffic flow confidentiality

In order to meet its goal and provide these services, IPsec relies on the AH and ESP headers as well as cryptographic key management protocols and procedures.

Depending on user, application, and system requirements IPsec employs an appropriate set of protocols to provide security services requested by a user or application. While a default set of algorithms and protocols is defined to support interoperability in the Internet, IPsec is sufficiently flexible for groups of individuals to define and use their own sets of algorithms. Such flexibility is imperative for successful deployment of this protocol suite so it can provide all requested services while not interfering with the network and its usability.

b. How IPsec Provides Desired Services

First of all, note that the IPsec architecture does not cover the implementation of specific encryption algorithms and other protocols, but it assumes that their implementation is secure. The best-designed security algorithm or protocol can fail if poorly implemented; so, while algorithm implementations are beyond the scope of the architecture, it is important to recognize that they play a crucial role in the effectiveness of IPsec.

An IPsec implementation relies on a Security Policy Database (SPD) for direction on how to treat IP packets. Based on the security policy laid out in the SPD, packets are either provided with security services, discarded, or allowed to bypass IPsec altogether. On a single host, IPsec allows the system to specify security protocols, and then determines the algorithms and cryptographic keys that will facilitate the selected services. Once the services are selected, the cryptographic keys must be created on the desired machines.

IPsec uses symmetric (shared secret) keys and Security Associations (SA). A SA is a "simplex 'connection' that affords security services to the traffic" [IPsec] that it carries. IPsec relies on a separate mechanism for distributing the cryptographic keys and managing the SAs. The Internet Security Association and Key Management Protocol (ISAKMP), specified in [ISAKMP], presents a framework for managing security associations and cryptographic keys. ISAKMP does not define any specific methods for managing and distributing keys. Instead, it sets guidelines that all IPsec key management protocols must obey. With this method, IPsec can rely on any key management mechanism that is based on the ISAKMP template. The Internet Key Exchange (IKE),

specified in [IKE], is an example of a public-key based approach for automatically distributing cryptographic keys. The keys may also be distributed manually or through some mechanism other than IKE. The distribution of keys, like encryption algorithms, is beyond the scope of [IPsec], so the design essentially assumes that effective key management and distribution methods are in use.

After key distribution, further communications between the involved nodes rely on the AH and ESP headers to provide the security services prescribed in the SPD. Both headers may provide connectionless integrity, data origin authentication, and an anti-replay service. The ESP can also provide confidentiality and limited traffic flow confidentiality.

2. QoSS

One way that QoSS functionality can be provided to the network is through IPsec. As discussed in [QoSS], QoSS functionality was added to OpenBSD's implementation of IPsec in IPv4.

a. Transitioning QoSS Capabilities to IPv6 in MYSEA

Implementing the QoSS capabilities in IPv6 will potentially involve changing the source code that implements it in IPv4. Since the concept was created and developed under IPv4, it is possible that some of the program code depends on peculiarities of that protocol. Such a situation would simply require "porting" those sections of code into conformance with IPv6. Otherwise, given the fact that IPsec is designed to function in either an IPv4 or an IPv6 environment, the QoSS additions to an IPsec implementation should be a transparent issue when switching protocols.

D. IPV4 VERSUS IPV6

Based on the above summaries of the IPv4 and IPv6 protocols, this section presents a comparison of the two designs. While some broad issues are addressed, this comparison primarily focuses on the issues that affect MYSEA. It seeks to pinpoint portions of the IPv6 design, if any, that could detract from the basic functionality that MYSEA aims to provide.

1. The Superior Design

The superiority of either design has yet to be quantitatively proven, and the focus of this discussion is not to attempt to discover which one is superior. IPv4 has been in use for over two decades, and since IPv6 is intended to be the next generation Internet Protocol, its design should contain beneficial alterations from the IPv4 design. These alterations are listed in the beginning of the discussion on IPv6 as the general changes to the IP design. Conversely, the IPv6 design alterations could potentially bring unwanted behaviors during implementation, or they could adversely affect MYSEA.

2. A Head to Head Comparison

Beginning with the IP headers, this section notes the changes and possible repercussions of those changes from IPv4 to IPv6. The IPv4 header is much smaller than the IPv6 header in terms of bytes; however, this size difference can be quickly narrowed when the IPv4 options field contains data. Since this options field has a variable length, the IPv4 header size is also variable, thereby adding to the complexity of header processing. On the other hand, the IPv6 header, while larger, has a static size. This stasis is a benefit to IPv6 because it reduces the possible complexity of the IP header. It also lessens the variability of overall packet size since the header size never changes. A further improvement for IPv6 is the fact that the design allows for more flexibility in sending optional information by using extension headers instead of imbedding the data in the IP header. This method makes it easier to define new options in the future while contributing to the modularity of the protocol.

As for addressing, the most obvious improvement in IPv6 is the massive increase in the size of the address space. This increase in IP addresses will mitigate concerns about the inability of IPv4 to provide enough addresses for the world. More subtly, the IPv6 addressing hierarchy will improve addressing efficiency throughout the Internet, thereby enhancing the efficiency of a distributed system such as MYSEA. While IPv4 uses CIDR to aggregate and improve Internet addressing, it is only a partial solution. There are many IPv4 addresses that originated before the implementation of CIDR. Consequently, the IPv4 addressing hierarchy is a heterogeneous model and is more complex than the IPv6 hierarchy.

The comparison of security features is, in a sense, unfair to IPv4. It was not initially designed with security capabilities, but it now has the capacity to provide authentication and encryption since IPsec introduced the capabilities. IPv6 was designed with the intent to include the ESP and AH headers as extension headers; so IPsec was essentially built into IPv6 while it was "patched" into IPv4. Again, from a security standpoint, the idea of embedding functionality has more appeal than functionality provided after the fact. Inclusion in the design from the outset lessens the chance of encountering undesirable "feature interactions" in the implementation.

From a design perspective, the functionality or service that IPsec provides does not differ based on which protocol is in use. However, due to the changes in IPv6, IPsec may prove more usable, more reliable, or it may provide more efficient security services in IPv6. Only quantitative analyses of the performance of IPsec in both protocols can provide these determinations. As for QoSS in MYSEA, a change from IPv4 to IPv6 should be transparent to its implementation.

3. Conclusion of the Comparison

It appears that the IPv6 design attempts to increase the overall modularity of the IP design. From the header to the extension headers to the aggregatable addressing hierarchy, the specifications for IPv6 appear to focus on modularizing the design while minimizing interdependencies of those modules. In general, modularity is good because it increases the flexibility of the design. Just as IPv6's modular header design makes it easier to define new options, modular components increase the ease of modifying single components without affecting the entire design.

Based on its design and its comparison with the IPv4 design, the conclusion is that IPv6 can at the least provide the same unaltered services as IPv4. Furthermore, IPv6 could possibly improve the efficiency and security of those services. Changes involving the addressing structure and the default MTU have the potential to provide added efficiency across the network; and the simplification of the design coupled with the fact that IPsec is part of the design can provide more assurance of security. IPv6's monumental address space should do away with the necessity for performing NAT in the

Internet; however, because its address hiding functionality is fundamental to the MYSEA design, that functionality must be implemented in an IPv6 version of MYSEA.

E. THE IPV4-TO-IPV6 TRANSITION

Over the last few years, a point of division has grown among the engineers and architects of the Internet. On one side of the debate stand those who believe that the shrinking address space of IPv4 (along with other concerns such as the size of routing tables) is not a significant problem. Opposing them are those who believe that IPv6 is the only option for the Internet's future communications protocol. Many among the IPv6 proponents believe that the immensely larger IPv6 address space will allay the world's IP address space concerns, and that the new protocol will greatly contribute to the advent of mobile and pervasive computing.

The obvious question arising from this debate is "who is right?" A potential answer could be that neither side is exclusively correct. As stated in [MECHS], "the Internet will need [both IPv4 and IPv6] compatibility for a long time … and perhaps indefinitely." Considering this possibility, it becomes clear that there is a need for mechanisms to allow seamless communication between nodes using either protocol. Therefore, this section does not seek to argue for one side or the other, but merely presents facts about current work intended to prepare the Internet for the use of IPv6. These transition mechanisms could also positively impact the use of MYSEA in an IPv6 environment.

1. Transition Mechanisms

The general transition mechanisms listed below are presented by Gilligan and Nordmark. They understand that "the key to a successful IPv6 transition is compatibility with the large installed base of IPv4 hosts and routers" [MECHS], so they defined the following standard mechanisms to provide such compatibility:

- Dual IP layer
- Configured tunneling of IPv6 over IPv4
- IPv4-compatible IPv6 addresses
- Automatic tunneling of IPv6 over IPv4

A machine with a dual IP layer, also referred to as a dual IP stack, has support for both Internet Protocols. Nearly every current PC computer system and router contains a dual IP stack, enabling it to communicate through either IPv4 or IPv6. Configured tunneling involves transmitting IPv6 data across an IPv4 link by encapsulating the IPv6 packets within IPv4 datagrams. A configured tunnel is a point-to-point connection, i.e., explicitly set up on a single link between two nodes. In automatic tunneling, a node uses an IPv4-compatible IPv6 address to automatically tunnel IPv6 packets over IPv4 networks. An IPv4-compatible address is an IPv6 address with an embedded IPv4 address. Addresses belonging to this group contain all zeros for their first ninety-six bits, and the following thirty-two bits represent the embedded IPv4 address. Figure 9 shows an example IPv4-compatible IPv6 address with sizes of the two major parts annotated.

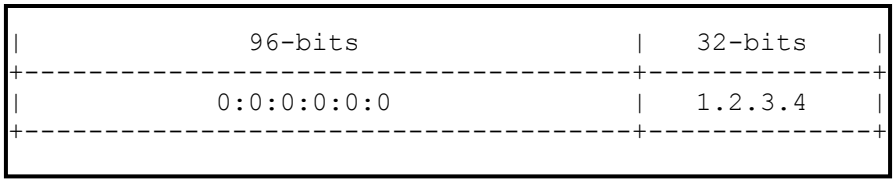

Figure 9. IPv4-compatible IPv6 Address [MECHS]

In addition to the mechanisms specified in [MECHS], the following transition mechanisms are defined in other documents:

- A mechanism for allowing communications between remote IPv6 domains that are separated by an IPv4 network, e.g., the Internet. This mechanism is known as 6to4, and is specified in [6to4].
- Network Address Translation - Protocol Translation (NAT-PT), a NAT mechanism that translates between IPv4 and IPv6, is specified in [NATPT].
- A mechanism for a dual IP layer machine that allows forwarding of transport-layer data between IPv6 and IPv4 networks. This mechanism is called a Transport Relay Translator (TRT) and is specified in [TRT].

The TRT is the higher-level cousin of NAT-PT. NAT and NAT-PT devices manipulate IP packets and IP addresses, modifying the addressing information before forwarding a packet on to its destination. A TRT functions one level above NAT-PT, forwarding TCP/UDP connections for specific services to a specified destination. The

TRT is based on transport relay (which can also be referred to as proxy) functionality. The 6to4 mechanism appears to currently be the most widely used transition mechanism. It is employed by host machines for users who wish to connect to the experimental IPv6 Internet backbone called the 6Bone.

The above tools are important for those users who wish to experiment with IPv6 while interacting with native IPv4 nodes. They also give experimenters the ability to access the IPv6 Internet backbone, called the 6bone, which is basically a group of interconnected IPv6 compliant machines. Users can tunnel their IPv6 traffic through their IPv4 Internet Service Providers (ISP), and the 6bone border routers will strip off the IPv4 headers and forward the IPv6 traffic to its destination.

2. Transition Mechanisms & MYSEA

Transition mechanisms are relevant to MYSEA because they can assist with the development of IPv6 solutions for the architecture. Theoretically, MYSEA can be distributed in such a way that a client workstation lies on a network separated from the server by one or more interconnected networks (i.e. the Internet). Assuming that MYSEA employs IPv6 and the networks between the server and client use IPv4, MYSEA would have to make use of the transition mechanisms in order function in the mixed environment.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION OF IPV6 MYSEA PROTOTYPE

This chapter contains a detailed description of the work involved in creating an IPv6 version of the MYSEA prototype. The general steps involved in this implementation were:

1. Set up the IPv4 MYSEA prototype test bed on three PCs.
2. Design an IPv6 solution with the functionality that MYSEA requires.
3. Upgrade protocol dependent software.
4. Enable IPv6 stacks on the appropriate MYSEA machines.
5. Create an IPv6 mechanism to perform the functionality of NAT.

The IPv4 implementation of the architecture served as a baseline for the work described here. After verification that it functioned properly in IPv4, the IPv6 implementation process could begin. The ensuing subsections will detail the design and implementation process beginning with the IPv4 prototype. Note that QoSS is not part of this implementation work.

A. IPV4 PROTOTYPE

1. Design

The current MYSEA prototype in IPv4 consists of two laptop PCs and a Compaq Pocket PC running the Linux OS. OpenBSD version 3.1 serves as both the server and client OSs; however, the server OS is modified to handle data labeling. All of the basic design issues are covered in the Chapter II discussion of MYSEA.

The subsections below discuss the differences between the "laptop" MYSEA prototype design and the "PC" prototype design.

a. Design Choices for the IPv4 Server

The server system remained unchanged from the previous prototype.

b. Design Choices for the IPv4 TPE

The TPE was implemented on the FreeBSD version 4.7 OS instead of the Linux OS used in the previous IPv4 prototype. FreeBSD was chosen as the TPE's OS for the following reasons: It has a very mature IPv6 networking stack, and it is generally a

very flexible OS. FreeBSD's flexibility could have provided many options during implementation of the TPE if necessary.

c. Design Choices for the IPv4 Client

As a testament to the flexibility of the MYSEA design, the client workstation runs the Windows XP OS while the current IPv4 prototype client runs OpenBSD. This difference in OSs had no effect on the function of MYSEA because the design allows for various client machines.

Other than the above minor changes and the use of different IP addresses, the "PC" prototype design was exactly the same as the "laptop" design. Figure 5 in Chapter II contains a diagram of the network setup, and it is reproduced here as Figure 10 for convenience.

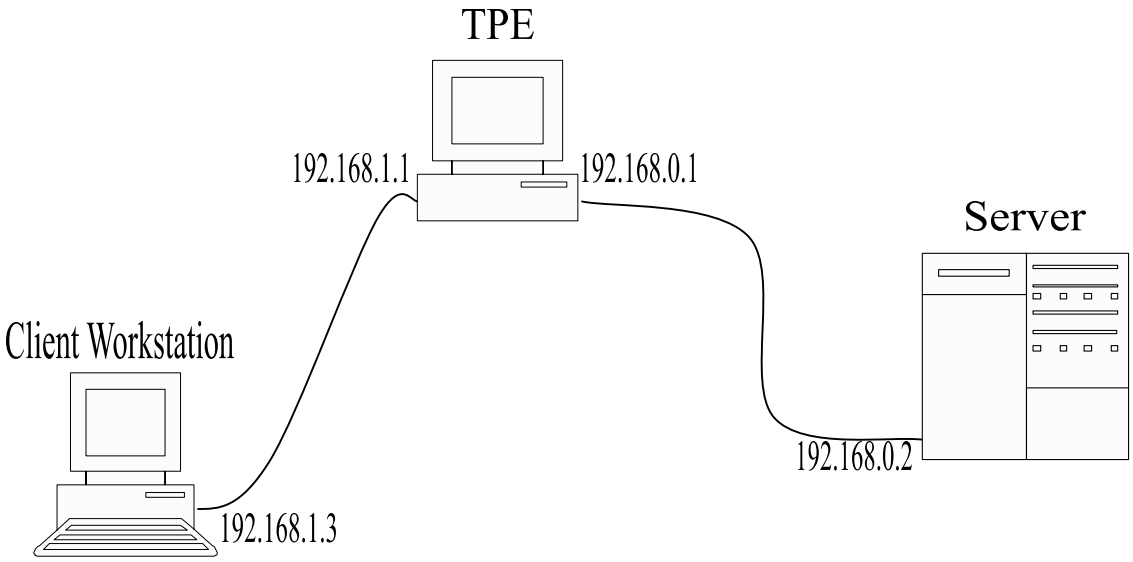

Figure 10. IPv4 MYSEA network diagram

2. Implementation

a. Setup of the IPv4 Server

The first step of the implementation of the IPv4 MYSEA prototype involved installation of the server's OS and completing the modifications to allow data labeling. Setting up the server also included modifying a file called "tcbe_list" to ensure

that it contained the TPE's IP address. Currently, the server uses this file to confirm the identities of the TPEs that have permission to communicate with it.

b. Setup of the IPv4 Trusted Path Extension (TPE)

Next, FreeBSD was installed as the OS for the TPE, and the Trusted Path Extension user interface software was loaded onto the machine. This software serves as the user's interface for authenticating himself to the server through the TPE.

c. Setup of the IPv4 Client

Windows XP was installed on the client machine, and it was connected to the TPE. No further actions were taken to prepare the client machine.

d. Verification of IPv4 MYSEA Functionality on Three PCs

Once communications capabilities were verified between the TPE and the other two machines, packet forwarding on the TPE was enabled and a packet capturing tool was used to verify that the client and server could communicate across the TPE.

With all communications verified, the final step was to enable NAT on the TPE. This step involved modifying the FreeBSD kernel to enable loading of firewall software upon startup, and then configuring the firewall to use its NAT functionality. The firewall called *ipf* and its NAT software, called *ipnat*, were used to provide the required functionality. Once NAT was enabled and functioning correctly, the IPv4 MYSEA network was complete and fully functional.

The all-PC prototype of MYSEA in IPv4 functioned exactly as the laptop prototype did. A demonstration of its functionality consisted of the following steps:

1. Execute the TPE user interface program.
2. Click the Secure Attention Request (SAR) key to initiate communications with the server.
3. Enter the login name of an authorized user of the system.
4. Enter the user's password.
5. Select the unclassified session level.
6. Enter the "Run" command.
7. On the client machine, open a Web browser.
8. Type the IP address of the server into the address bar.
9. When the web page opens, click the link to access unclassified information.
10. Click the "Back" button on the browser.
11. Click the link to the classified information -- no useful information appears.

12. Click “Back” on the browser again.
13. Return to the TPE, click the SAR key, and select the classified session level.
14. Enter the “Run” command.
15. Return to the client machine and click the link to access classified information.
16. View classified information.
17. Click the “Back” button.
18. Click the link to the unclassified information and view it to verify read-down capability

Running the demonstration provided confirmation of a functioning IPv4 MYSEA prototype on three PCs that required no major modifications. Given this working implementation, one could then assume that any modifications to the design or implementation of the architecture in IPv6 were attributable to the use of the new protocol.

B. IPV6 PROTOTYPE

1. Design

The basic design of the all-PC IPv6 MYSEA prototype is no different from the IPv4 version of the architecture. The OSs of the machines remained the same as they are in the all-PC IPv4 prototype. The server’s OS has remained static throughout this project; the client’s OS can be any available PC operating system; and the TPE OS was chosen for this project before the design and implementation of the all-PC IPv4 MYSEA prototype, so it too remains unchanged.

The TPE’s OS, FreeBSD 4.7, contained an IPv6 stack that appeared to be one of the most mature and robust IPv6 stacks in development. Also, as stated above, the flexibility of FreeBSD was likely to aid in this work. Naturally, each interface in the IPv6 network has a different IP address than the interfaces in the IPv4 network setup. Additionally, bear in mind that the link from the client to the TPE could be an IPv4 link without affecting the function of the architecture. In fact, it does not matter how the client and TPE communicate as long as the TPE has a mechanism for translating communications to the server network. Since IP is the standard network protocol, available in all COTS client systems, it is a reasonable choice for this link. Figure 11 shows the design of the IPv6 MYSEA prototype.

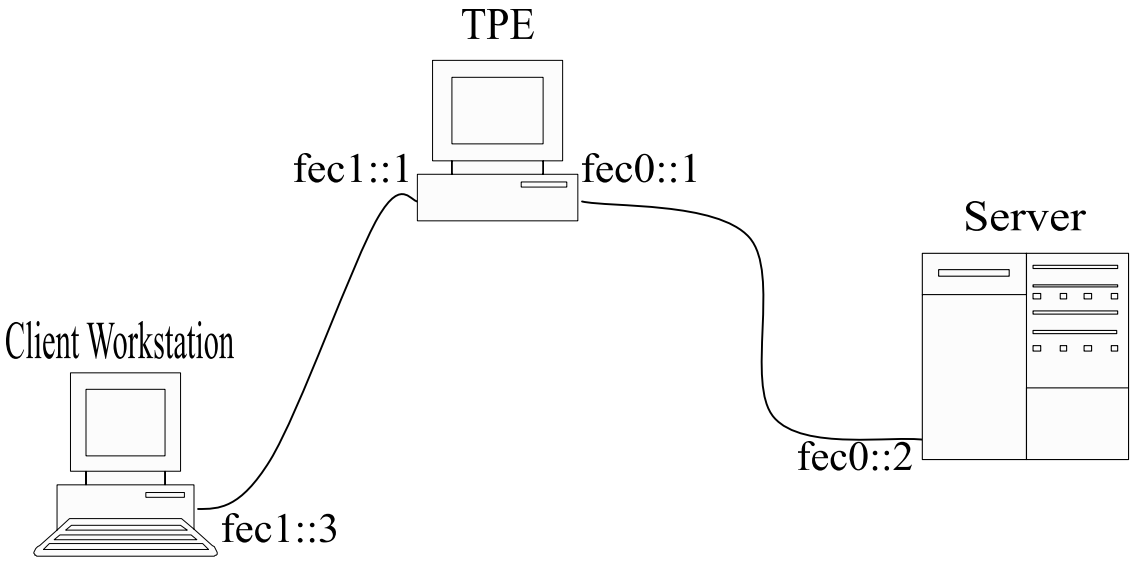

Figure 11. IPv6 MYSEA network diagram

2. Implementation

The information below provides the details of the steps taken to implement the MYSEA architecture on an IPv6 network. Aside from having to update the applications that were not compatible with IPv6, the implementation was very similar to that for IPv4 up to the following point: The provision of the NAT functionality on the TPE required the greatest amount of effort because IPv6 lacks native NAT functionality. The following list provides a basic overview of the steps taken to complete the implementation:

- Ensure installation of IPv6 networking stacks on all machines.
- Verify communications abilities between the TPE and server, and between the TPE and client.
- Enable IPv6 packet forwarding on the TPE and verify communications abilities across the TPE.
- Ensure IPv6-compatible application software is installed where necessary.
- Update MYSEA-specific software on the server and TPE.
- Decide on a mechanism to provide the necessary NAT functionality on the TPE.
- Install or implement this mechanism.
- Run the MYSEA demonstration and verify correct function of the architecture.

a. Enable Network Communications

The client, running Windows XP, was the only machine in the IPv6 design that did not have its IPv6 stack enabled by default. Simply typing "ipv6 install" at the command prompt enabled IPv6 functionality on the client machine. Once IPv6 capabilities were verified in all PCs, the *ping6* utility was used to verify that the machines could communicate in IPv6. Then forwarding of IPv6 packets was enabled on the TPE, and it was verified that the server and client could communicate across the TPE. That step completed the enabling of communications in the network. The next large step in the process entailed ensuring that all applications and programs could function in an IPv6 environment.

b. Port Applications to IPv6

Many of the applications used in the original prototype depended on IPv4 as the communications protocol, and minor changes had to be made to the MYSEA-specific programs on the server. These programs included a modified version of *inetd*, which handled incoming connection requests, and the trusted path services program that handled authentication from the TPE. The TPE user interface program also had to be modified to use IPv6 for the authentication process. Additionally, the server's Web server application had to be replaced with one that functioned in an IPv6-enabled environment; however, it turned out that the client's Web browser functioned with IPv6 by default. With sound communications and IPv6-ready applications, the final step in this implementation was to provide the NAT functionality that would in effect hide the true IP address of the client from the server.

c. Considerations before Implementing NAT Functionality in IPv6

Before attempting creation of a NAT mechanism on the TPE from scratch, other options were explored. As mentioned in Chapter III, section E, there is a specification for a mechanism called NAT-PT. This mechanism would give a legacy IPv4 client machine the ability to access an IPv6 MYSEA network. Unfortunately, there were apparently no reliable implementations of NAT-PT available to be incorporated into MYSEA. After reaching this stage, there was no point in considering the use of a hybrid network design. It made more sense to implement NAT in IPv6 than to implement the

more complex NAT-PT. The decision was made to implement the client-to-TPE link in IPv6, resulting in the diagram in Figure 11.

The possibility of taking advantage of the flexibility in IPv6's addressing architecture was also explored. Recall that every machine must have a Link-local IP address. Even though these addresses were not used, and therefore not shown in the network diagram, each machine did indeed have a Link-local address through which it could communicate. After much review and consideration of hypothetical designs, there appeared to be no effective way to use the addressing architecture, or other unique concepts such as load-sharing, to provide the needed functionality without still placing a subset of NAT functionality on the TPE.

Coming to this conclusion meant that a new mechanism must be created that will allow the TPE to provide the same functions that NAT provided to MYSEA in IPv4. However, rather than creating an IPv6 NAT mechanism from scratch, it made more sense to find and use some existing open-source software as a sort of template. The existing IPv4 NAT implementations proved far too large and complex to create a port for IPv6, especially since MYSEA only requires the address replacement functionality and current NAT implementations provide much more. For instance, NAT mechanisms are expected to be able to perform port address translation (PAT), and they have the functionality to create mappings for ranges of IP addresses. PAT involves mapping a port on the NAT device to the IP address of an internal node. See [NAT_TERM] and [NAT] for more intricacies of NAT and PAT. That extra functionality complicates any attempt to port a subset of NAT to IPv6. The few thousand lines of source code that are separated into multiple files are very interdependent, and the program flow is extremely complex. Furthermore, NAT code runs in kernel space and uses kernel-specific operations; therefore, a port of NAT to IPv6 would be OS dependent and most likely only function on the FreeBSD kernel in this case. Then, in the future, developers would have to either completely rebuild or rewrite parts of the code for it to be compatible with the final TPE kernel. The situation was as follows: NAT-PT implementations were unavailable, the use of IPv4-mapped addresses would still require extra functionality on the TPE, and there appeared to be no implemented TRTs that could be used.

However, there was a user-level program called *faithd* (faith dee) available for the FreeBSD OS that somewhat performed the functionality needed to complete the IPv6 MYSEA implementation. Though it is not stated in any of the program documentation, *faithd* is actually a TRT that uses IPv4-mapped IPv6 addresses to give an IPv6 node the ability to access an IPv4 network. This functionality is the opposite of what the TPE would need to perform if the client-to-TPE link was an IPv4 link. Still, the code for *faithd* was enough to serve as a partial template for a pure IPv6 transport-layer connection forwarder. More importantly, *faithd* was packaged with the code that performs relaying of TCP and UDP connections, so the program only required modifications to its calling interface. Even though the final version of this forwarder would not perform true NAT because it operates on the transport layer instead of the IP layer (making it more like a proxy), it does provide the functionality necessary for MYSEA to function in an IPv6 environment.

d. Implementation of IPv6 NAT Functionality

Since *faithd* was designed to provide service to IPv6 nodes, it contained the information necessary for handling IPv6 connections. Instead of forwarding an IPv6 connection to an IPv4 connection, the modified *faithd* (called *myseatr*) would forward data from a specific IPv6 connection (the client) to another specific IPv6 connection (the server).

The first step in changing the functionality of *faithd* was to review the code to gain an understanding of how it worked. The code contained roughly one thousand lines, so this goal was not unattainable. After virtually mapping the source code, the socket calls and connection operations had to be adjusted to use only pure IPv6 interfaces and addresses. All code involving IPv4 was removed from the program. The program's ability to run from *inetd* was also removed for simplicity. Appendix A contains the source code of the program.

myseatr functions as follows: at the start of program execution, *myseatr* prompts the user for the interface names on the client and server networks, and for the client and server IPv6 addresses. It uses the interface names to ensure that it only forwards connections from the client interface. Also, the client's IP address is the only

address from which the program will forward data, and the server's IP address is always the destination. When the client machine makes a TCP connection to the machine on which *myseatr* is running, it accepts that connection, makes an identical TCP connection to the server, and forwards ensuing data between the client and server connections. Upon completion of the program modifications, the demonstration was run and the functionality of MYSEA in IPv6 was verified. However, since *myseatr* requires a direct TCP connection from the client, the demonstration had to be modified accordingly. Instead of typing the server's IP address into the browser's address bar, the user must type in the IP address of the TPE. The following section explores the implications of using *myseatr* to provide the required functionality.

3. Implications of Performing Proxy Translation Vice True NAT

Implementing MYSEA's NAT functionality through what is essentially an IPv6 proxy gets the job done, but it could have a major impact on the rest of the architecture. This section explores the effects that a proxy translator could have on the MYSEA network.

First of all, the proxy method is misleading because the user of the client machine would have to connect to the TPE for all services, and the TPE would forward those connections to the appropriate MYSEA servers. Even though this situation has an impact on the use of MYSEA, it should not have any adverse effects on the function of MYSEA.

With regard to the connection, a proxy breaks the end-to-end connection from the client to the server. However, since the client and TPE are considered by the server to be one entity, the connection only needs to be end-to-end between the TPE and server. Note also that true NAT disrupts end-to-end communications.

Another drawback to transport layer connection forwarding is the fact that the forwarder only recognizes connections to specific services such as web, telnet, and file transfer protocol (FTP). This means that an instance of the *myseatr* program can forward one of these services and only one of them. A simple rectification to this problem would be to either run *myseatr* from the Internet daemon (*inetd*) – i.e., restore *myseatr*'s ability to run from inetd – or add *inetd*-like functionality to *myseatr*.

The fact that this TPE implementation is a proxy means that it is more constrained in the potential operations it can perform, and it is much less flexible than NAT. As stated in the above paragraph, the service it supports must be explicitly stated, but a true NAT device deals with packets and pays no attention to higher layer protocols. NAT also provides more granularity and flexibility in its abilities to manipulate communications. Of course, this flexibility comes with the price of requiring more expertise and energy to implement NAT in a UNIX environment. This fact might have been an advantage to the proxy method if the TPE had already been developed in a UNIX environment. However, the objective is to run the TPE in a high assurance, non-UNIX environment; therefore, until such an environment becomes available, one can draw no conclusions about the relative time and energy required to build the TPE with the necessary NAT functionality versus the time and energy needed for the proxy functionality.

V. CONCLUSIONS AND FUTURE WORK

The paragraphs below briefly restate the conclusions from the comparison of the IPv4 and IPv6 designs. Following that discussion is a section containing future work beyond this research.

A. THE FACTS ABOUT MYSEA AND IP

MYSEA depends on the Internet Protocol for communications throughout the distributed network. Currently, IPv4 is the standard in use, but since IPv6 is a viable replacement, its use and how it will integrate into MYSEA should be explored. As evidenced by the existing MYSEA prototype, IPv4 has the capabilities to provide the necessary services that MYSEA requires; and even though IPsec was patched into the protocol, it can effectively provide the necessary security services. IPv6, with its design improvements, may provide improved IP and IPsec services to MYSEA.

1. Concerns with IPv4

First, IPv4 has been in use for greater than twenty years. While supporting the fact that IPv4 was well designed and has proven more useful than originally intended, this fact also serves as a reminder, along with the other concerns mentioned in this paper, that the Internet needs an upgrade. The IPv4 address space, while large in its own right, does not have the capacity to handle the IP addressing needs of the world either today or in the future. IPv4 addressing and Internet routing tables are other sources of concern about the protocol. Finally, IPsec was a functionality add-on to provide security for IPv4. This fact serves to lessen the assurance that IPv4 can provide concrete authentication and encryption for IP datagrams.

2. The Potential within IPv6

a. Design

The IPv6 design does contain improvements over the IPv4 design, and some of these improvements are discussed here.

IPv6 has a more structured overall design, and the IPv6 header has been streamlined. Its simplification has resulted in a static header size, another improvement.

Also as a result of the simplification of the header, IPv6 allows for extension headers that are used to encode optional data. This method of delivering optional data adds flexibility to the protocol by allowing for new options and headers to be easily defined and implemented in the future. The IPv6 design adds structure to the use of these headers by recommending an order for including these headers in a packet.

b. Addressing

Perhaps the most obvious improvement in IPv6 is the increase in the size of the address space and the way the addressing hierarchy is set up. The addressing hierarchy is intended to allow better aggregation of IP addresses and to take full advantage of the benefits that CIDR provides to IPv4.

c. Security

On the subject of security, IPv6 was built with native support for IPsec. Unlike IPv4, where IPsec was not built into the design, IPv6 incorporates the IPsec headers as extension headers. This innate integration of IPsec can provide more assured security services for MYSEA. Moreover, it supports the security practice of maintaining simplicity in order to provide greater assurance that security is maintained.

d. QoSS

The proof of concept implementation of QoSS involved the modification of part of an IPsec implementation. Given that IPsec is the chosen avenue for the ultimate provision of QoSS capabilities in MYSEA, then the transition to IPv6 will have no visible effect on those services.

3. The Impending Transition and MYSEA

Already in the earliest stages, the world will begin a gradual transition from IPv4 to IPv6. The tools that have been defined and developed to aid in this transition, as well as tools that have yet to be developed, have the potential to aid in the deployment of the distributed MYSEA architecture. These tools could also aid in MYSEA's own transition to IPv6 by allowing a legacy IPv4 client to access an IPv6 MYSEA network or vice versa.

B. FUTURE WORK

The following sections introduce items of future work beyond that performed in this thesis.

1. Solve the NAT Problem

One area of possible work for MYSEA in IPv6 is the task of finding a native IPv6 alternative to implementing NAT. Such work could include exploring for and possibly finding an IPv6 workaround for NAT. If the conclusion of this work is that MYSEA in IPv6 must depend on NAT, then another possible future project is to develop the specific NAT functionality in IPv6 on which MYSEA depends.

Another project involving NAT would include a comparison of performing true NAT versus transport relay in MYSEA. Each method comes with different pros and cons, and MYSEA will likely benefit from one method more than the other.

Finally, for NAT in MYSEA, the following situation can be explored: since every IPv6 conformant client boots up with a unique link-local address, the step for assigning it a site-local address can be skipped altogether. The link-local address will allow the client to communicate with the TPE, but packets containing that address can never exist beyond that link. However, the TPE can run NAT such that it replaces the client's link-local address with its own (valid beyond the client-TPE link), and then forward the resulting packets. Such an implementation will restrict the client's communications ability, thereby requiring nothing beyond a conformant IPv6 implementation on the TPE, and it totally supports the MYSEA design.

2. Research on a Hybrid IPv4 and IPv6 Network

One can explore the viability of running MYSEA as a hybrid IPv4 and IPv6 network. The possible gains and losses of such a MYSEA implementation can be addressed during that exploration. Such work can also include a hybrid prototype implementation of MYSEA that runs NAT-PT on the TPE.

3. Technical Work

A future project that is more technical than those presented above involves a quantitative analysis of network performance for networks using IPv4 versus Pv6. Such an analysis can be performed on a MYSEA implementation, and can serve as concrete

evidence of which protocol benefits MYSEA more. In addition, an analysis can be performed on the performance of IPsec in either protocol. This analysis will address such items as overall usability, ease of configuration, and general performance of service (e.g. change in transfer rate due to use of the ESP header).

4. Work on QoSS

Implementation and analysis of the provision of QoSS in IPv6 is another future project involving MYSEA and IPv6. Research can be performed on the best way to provide QoSS in an IPv6 environment, and further work can be done on its effectiveness in such an environment.

LIST OF REFERENCES

[3COM] 3Com Corporation. "Understanding IP Addressing: Everything You Ever Wanted To Know." [http://www.3com.com/other/pdfs/infra/corpinfo/en_US/501302.pdf]. April 2003.

[QoSS] Agar, C.D., and others "IPsec Modulation for Quality of Security Service." Proceedings of the International System Security Engineering Association Conference, Orlando Florida. 13 March 2002.

[SUBVERSION] Anderson, Emory. *A Demonstration of the Subversion Threat: Facing a Critical Responsibility in the Defense of Cyberspace.* Master's Thesis. Naval Postgraduate School. Monterey, California. March 2002.

[PROBLEM] Bradner, S. "The Problem of Being Permanent." [http://www.nwfusion.com/archive/1995/95-09-04the_-a.html]. September 1995.

[REC_IPng] Bradner, S., and Mankin, A. "The Recommendation for the IP Next Generation Protocol." RFC 1752. January 1995.

[6to4] Carpenter, B., and Moore, K. "Connection of IPv6 Domains via IPv4 Clouds," RFC 3056. February 2001.

[CISCO] Cisco Systems. "IP Addressing and Subnetting for New Users." [http://www.cisco.com/warp/public/701/3.html]. Also available in pdf format. [http://www.cisco.com/warp/public/701/3.pdf]. April 2003.

[IP6] Deering S., and Hinden R. "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460. December 1998.

[IP6 ADDR] Deering S., and Hinden R. "IP Version 6 Addressing Architecture," RFC 2373. July 1998.

[MECHS] Gilligan, R., and Nordmark, E. "Transition Mechanisms for IPv6 Hosts and Routers," RFC 2893. August 2000.

[TRT] Hagino, J., and Yamamoto, K. "An IPv6-to-IPv4 Transport Relay Translator," RFC 3142. June 2001.

[IKE] Harkins, D. and Carrel, D. "The Internet Key Exhange (IKE)," RFC 2409. November 1998.

[QoSS] Irvine, C., and Levin, T., "Quality of Security Service," paper presented at Proceedings of the New Security Paradigms Workshop. Bollycotton, Ireland. 18 - 22 September 2000.

[MYSEA ARCH] Irvine, C., and others. "MYSEA Security Architecture," Naval Postgraduate School, May 2002.

[MYSEA PROJ] Irvine, C., and others. "MYSEA Security Enhanced Architecture Project," Naval Postgraduate School, April 2003.

[IPsec] Kent, S. and Atkinson, R. "Security Architecture for the Internet Protocol," RFC 2401. November 1998.

[ISAKMP] Piper, D. "The Internet IP Security Domain of Interpretation for ISAKMP," RFC 2407. November 1998.

[IP] Postel, J. "Internet Protocol," RFC 791, University of Southern California / Information Sciences Institute. September 1981.

[NAT] Srisuresh, P., and Egevang, K. "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022. January 2001.

[NAT_TERM] Srisuresh, P., and Holdrege, M. "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663. August 1999.

[MEMO] Department of Defense memorandum, SUBJECT: Internet Protocol version 6 (IPv6). June 9, 2003.

[DOCMAP] Thayer, R., Doraswamy, N., and Glenn, R. "IP Security Document Roadmap," RFC 2411. November 1998.

[NATPT] Tsirtsis, G., and Srisuresh, P. "Network Address Translation – Protocol Translation (NAT-PT)," RFC 2766. February 2000.

APPENDIX

```
/*    MRO: This code is based on the version of faithd given below */

/*    $KAME: faithd.c,v 1.46 2002/01/24 16:40:42 sumikawa Exp $   */

/*
 * Copyright (C) 1997 and 1998 WIDE Project.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 * 3. Neither the name of the project nor the names of its contributors
 *    may be used to endorse or promote products derived from this
 *    software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS ``AS IS''
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
 * TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE PROJECT OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 */

/*
 * MRO: This program is a user level tcp connection forwarder for IPv6.
 *
 * Usage: myseatr [<port> <progpath> <arg1(progname)> <arg2> ...]
 *    e.g. myseatr -d http
*/

#include <sys/param.h>
#include <sys/types.h>
#include <sys/sysctl.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <libutil.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <syslog.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include <fcntl.h>
#include <termios.h>

#include <net/if_types.h>
#include <net/if.h>
#include <net/route.h>
#include <net/if_dl.h>

#include <netinet/in.h> /* MRO: includes netinet6/in6.h */
#include <arpa/inet.h>
#include <netdb.h>
#include <ifaddrs.h>

#include "faithd.h"
#include "prefix.h"

char *serverpath = NULL;
char *serverarg[MAXARGV + 1];
static char *myseatrname = NULL;
char logname[BUFSIZ];
char procname[BUFSIZ];

struct myaddrs {
      struct myaddrs *next;
      struct sockaddr *addr;
};

struct myaddrs *myaddrs = NULL;
static const char *service;
static int sockfd = 0;
int dflag = 0;
static int pflag = 0;
static int inetd = 0;
static char *configfile = NULL;

/*************************************************************************
*     MRO: following are my global variables.
*************************************************************************/
/* mySockAddr stores addresses useful. */
struct mySockAddr {
      /* my socket address on server side */
      struct sockaddr_in6 tpe_s;

      /* client's socket address when connected */
      struct sockaddr_in6 clientAddr;

      /* Socket to listen on */
      struct sockaddr_in6 tpe_c; };
```

```
struct mySockAddr *mySock;

char ifServ[5];
char ifClient[5];
char serverAddr[NI_MAXHOST];
char clientAddr[NI_MAXHOST];

/*********************************************************************
*     MRO: Function prototypes, modified
*********************************************************************/

int main __P((int, char **));

static int daemon_main __P((int, char **));

static void play_service __P((int));

static void play_child __P((int, struct sockaddr *));

static int fwd_choice __P((struct sockaddr_in6 *));

static int subSrcAddr __P((struct sockaddr_in6 *));

static void grab_myaddrs __P((void));

/*********************************************************************
*     MRO: Function prototypes, unmodified
*********************************************************************/

static void sig_child __P((int));

static void sig_terminate __P((int));

static void start_daemon __P((void));

static void exit_stderr __P((const char *, ...))
      __attribute__((__format__(__printf__, 1, 2)));

static void update_myaddrs __P((void));

static void usage __P((void));

/*********************************************************************
* MRO
* Function:       main
* Description:    The main function.  Currently, it always calls
*                 daemon_main, but it can possibly call an inetd_main
*                 if called from inetd.
*********************************************************************/
int
main(int argc, char **argv)
{

      /* MRO: Get network specific information from user */

      printf("Type the TPE's server-side interface name: ");
```

```
	scanf("%s",ifServ);

	printf("Type the TPE's client-side interface name: ");
	scanf("%s",ifClient);

	printf("\nInput the server's IPv6 address: ");
	scanf("%s", serverAddr);

	printf("Input the client's IPv6 address: ");
	scanf("%s", clientAddr);

	/*
	 * Initializing stuff
	 */

	/* MRO: have to malloc for space */
	mySock = (struct mySockAddr *)malloc(sizeof(struct mySockAddr));
	if (!mySock) {
		exit_failure("not enough core");
		//NOTREACHED
	}

	myseatrname = strrchr(argv[0], '/');
	if (myseatrname)
		myseatrname++;
	else
		myseatrname = argv[0];

	return daemon_main(argc, argv);
}

/*****************************************************************
* MRO
* Function:	daemon_main
* Description:	This function is called by the main function when
*		the program is run as a daemon.  It performs the same
*		initial operations that the program would as a daemon.
*****************************************************************/
static int
daemon_main(int argc, char **argv)
{
	struct addrinfo hints, *res, *myres;
	struct sockaddr_in6 *tempSin6;
	int s_wld, error, i, serverargc, on = 1;
	int family = AF_INET6;
	int c;

	while ((c = getopt(argc, argv, "df:p46")) != -1) {
		switch (c) {
		case 'd':
			dflag++;
			break;
		case 'f':
			configfile = optarg;
			break;
		case 'p':
			pflag++;
```

```
                break;
#ifdef FAITH4
            case '4':
                family = AF_INET;
                break;
            case '6':
                family = AF_INET6;
                break;
#endif
            default:
                usage();
                /*NOTREACHED*/
            }
        }
        argc -= optind;
        argv += optind;

        grab_myaddrs();

        switch (argc) {
        case 0:
            usage();
            /*NOTREACHED*/
        default:
            serverargc = argc - NUMARG;
            if (serverargc >= MAXARGV)
                exit_stderr("too many arguments");

            serverpath = malloc(strlen(argv[NUMPRG]) + 1);
            strcpy(serverpath, argv[NUMPRG]);
            for (i = 0; i < serverargc; i++) {
                serverarg[i] = malloc(strlen(argv[i + NUMARG]) + 1);
                strcpy(serverarg[i], argv[i + NUMARG]);
            }
            serverarg[i] = NULL;
            /* fall through */
        case 1:     /* no local service */
            service = argv[NUMPRT];
            break;
        }

        /*
         * Opening socket for this service.
         */

        memset(&hints, 0, sizeof(hints));
        hints.ai_flags = AI_PASSIVE;
        hints.ai_family = family;
        hints.ai_socktype = SOCK_STREAM;
        hints.ai_protocol = 0;

        error = getaddrinfo(NULL, service, &hints, &res);
        if (error)
            exit_failure("getaddrinfo: %s", gai_strerror(error));

        s_wld = socket(res->ai_family, res->ai_socktype,
                res->ai_protocol);
```

```
if (s_wld == -1)
      exit_failure("socket(s_wld): %s", strerror(errno));

error = setsockopt(s_wld, SOL_SOCKET, SO_REUSEADDR, &on,
            sizeof(on));
if (error == -1)
      exit_failure("setsockopt(SO_REUSEADDR): %s",
            strerror(errno));

error = setsockopt(s_wld, SOL_SOCKET, SO_OOBINLINE, &on,
            sizeof(on));
if (error == -1)
      exit_failure("setsockopt(SO_OOBINLINE): %s",
            strerror(errno));

error = bind(s_wld, (struct sockaddr *)res->ai_addr,
      res->ai_addrlen);
if (error == -1) {
      switch (errno) {
            case EAGAIN:
                  syslog(LOG_WARNING, "error: EAGAIN");
                  break;
            case EBADF:
                  syslog(LOG_WARNING, "error: EBADF");
                  break;
            case ENOTSOCK:
                  syslog(LOG_WARNING, "error: ENOTSOCK");
                  break;
            case EADDRNOTAVAIL:
                  syslog(LOG_WARNING, "error: EADDRNOTAVAIL");
                  break;
            case EADDRINUSE:
                  syslog(LOG_WARNING, "error: EADDRINUSE");
                  break;
            case EACCES:
                  syslog(LOG_WARNING, "error: EACCES");
                  break;
            case EFAULT:
                  syslog(LOG_WARNING, "error: EFAULT");
                  break;
            default:
                  break;
      } //switch
      exit_failure("bind: %s", strerror(errno));
}

error = listen(s_wld, 5);
if (error == -1)
      exit_failure("listen: %s", strerror(errno));

sockfd = socket(PF_ROUTE, SOCK_RAW, PF_UNSPEC);
if (sockfd < 0) {
      exit_failure("socket(PF_ROUTE): %s", strerror(errno));
      /*NOTREACHED*/
}
```

```
	/*
	 * Everything is OK.
	 */

	start_daemon();

	snprintf(logname, sizeof(logname), "myseatr %s", service);
	snprintf(procname, sizeof(procname), "accepting port %s",
	      service);
	openlog(logname, LOG_PID | LOG_NOWAIT, LOG_DAEMON);
	syslog(LOG_INFO, "Starting myfaith daemon for %s port", service);

	freeaddrinfo(res);
	play_service(s_wld);

	/* NOTREACHED */
	exit(1);    /*pacify gcc*/

} //end daemon_main

/*************************************************************************
* MRO
* Function:       play_service
* Description:    Waits for and accepts connections, and then forks to
*                 allow child process to handle the connections.  The
*                 forked process calls play_child after forking.
*************************************************************************/
static void
play_service(int s_wld)
{
	struct sockaddr_in6 srcaddr;
	int len;
	int s_src;
	pid_t child_pid;
	fd_set rfds;
	int error;
	int maxfd;

	/*
	 * Wait, accept, fork, faith....
	 */
again:
	setproctitle("%s", procname);

	FD_ZERO(&rfds);
	FD_SET(s_wld, &rfds);
	maxfd = s_wld;

	if (sockfd) {
	      FD_SET(sockfd, &rfds);
	      maxfd = (maxfd < sockfd) ? sockfd : maxfd;
	}

	error = select(maxfd + 1, &rfds, NULL, NULL, NULL);
	if (error < 0) {
```

```
            if (errno == EINTR)
                  goto again;
            exit_failure("select: %s", strerror(errno));
            /*NOTREACHED*/
      }

      if (FD_ISSET(sockfd, &rfds)) {
            update_myaddrs();
      }

      if (FD_ISSET(s_wld, &rfds)) {
            len = sizeof(srcaddr);
            s_src = accept(s_wld, (struct sockaddr *)&srcaddr,
                  &len);
            if (s_src < 0) {
                  if (errno == ECONNABORTED)
                        goto again;
                  exit_failure("socket(accept): %s", strerror(errno));
                  /*NOTREACHED*/
            }

            /* MRO: get the client's address */
            memcpy(&mySock->clientAddr, &srcaddr, srcaddr.sin6_len);

            child_pid = fork();

            if (child_pid == 0) {
                  /* child process */
                  close(s_wld);
                  closelog();
                  openlog(logname, LOG_PID | LOG_NOWAIT, LOG_DAEMON);
                  play_child(s_src, (struct sockaddr *)&srcaddr);
                  exit_failure("should never reach here");
                  /*NOTREACHED*/
            }
      else {
                  /* parent process */
                  close(s_src);
                  if (child_pid == -1)
                        syslog(LOG_ERR, "can't fork");
            }
      }
      goto again;
} //end play_service

/*********************************************************************
* MRO
* Function:       play_child
* Description:    Calls the functions that ensure the connection should
*                 be forwarded, then calls the tcp relay function.
*********************************************************************/
static void
play_child(int s_src, struct sockaddr *srcaddr)
{
      struct sockaddr_in6 dstaddr6; /* destination address */
      struct sockaddr_in6 *sa;
      char sad[NI_MAXHOST];
```

```
char src[NI_MAXHOST];
char dst6[NI_MAXHOST];
int len = sizeof(dstaddr6);
int s_dst, error, hport, nresvport, on = 1;
struct timeval tv;

struct addrinfo hints, *res;

tv.tv_sec = 1;
tv.tv_usec = 0;

getnameinfo(srcaddr, srcaddr->sa_len,
      src, sizeof(src), NULL, 0, NI_NUMERICHOST);
syslog(LOG_INFO, "accepted a client from %s", src);

error = getsockname(s_src, (struct sockaddr *)&dstaddr6, &len);
if (error == -1) {
      exit_failure("getsockname: %s", strerror(errno));
      /*NOTREACHED*/
}

getnameinfo((struct sockaddr *)&dstaddr6, len,
      dst6, sizeof(dst6), NULL, 0, NI_NUMERICHOST);
syslog(LOG_INFO, "the client is connecting to %s", dst6);

/* MRO: for now, the destination is always the server */
memset(&hints, 0, sizeof(hints));
  hints.ai_flags = AI_NUMERICHOST;
  hints.ai_family = AF_INET6;
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_protocol = 0;

  error = getaddrinfo(serverAddr, service, &hints, &res);
  if (error)
          exit_failure("getaddrinfo: %s", gai_strerror(error));

sa = (struct sockaddr_in6 *)res->ai_addr;

getnameinfo((struct sockaddr *)sa, sa->sin6_len,
          sad, sizeof(sad), NULL, 0, NI_NUMERICHOST);

/* MRO: if substitution fails, don't forward connection */
if (!fwd_choice((struct sockaddr_in6 *)srcaddr)) {
      close(s_src);
      exit_success("Did not forward connection");
} //end if

/*
 * Forward the connection
 */

syslog(LOG_INFO, "The forwarder is connecting to %s", sad);

setproctitle("child on port %s, %s -> %s", service, src,
            serverAddr);
```

```
hport = ntohs(((struct sockaddr_in6 *)&dstaddr6)->sin6_port);

switch (hport) {
case RLOGIN_PORT:
case RSH_PORT:
      s_dst = rresvport_af(&nresvport, sa->sin6_family);
      break;
default:
      if (pflag)
            s_dst = rresvport_af(&nresvport, sa->sin6_family);
      else
            s_dst = socket(res->ai_family, res->ai_socktype,
                        res->ai_protocol);
      break;
}
if (s_dst < 0) {
      exit_failure("socket(s_dst): %s", strerror(errno));
      /*NOTREACHED*/
}

error = setsockopt(s_dst, SOL_SOCKET, SO_OOBINLINE, &on,
            sizeof(on));
if (error < 0) {
      exit_failure("setsockopt(SO_OOBINLINE): %s",
            strerror(errno));
      //NOTREACHED
}

error = setsockopt(s_src, SOL_SOCKET, SO_SNDTIMEO, &tv,
            sizeof(tv));
if (error < 0) {
      exit_failure("setsockopt(SO_SNDTIMEO): %s",
            strerror(errno));
      //NOTREACHED
}
error = setsockopt(s_dst, SOL_SOCKET, SO_SNDTIMEO, &tv,
            sizeof(tv));
if (error < 0) {
      exit_failure("setsockopt(SO_SNDTIMEO): %s",
            strerror(errno));
      //NOTREACHED
}


error = connect(s_dst, res->ai_addr, res->ai_addrlen);
if (error < 0) {
      switch (errno) {
                  case EAFNOSUPPORT:
                        syslog(LOG_WARNING,
                            "error: EAFNOSUPPORT");
                        break;
                  case EBADF:
                        syslog(LOG_WARNING, "error: EBADF");
                        break;
                  case ENOTSOCK:
                        syslog(LOG_WARNING, "error: ENOTSOCK");
                        break;
```

```
                        case EADDRNOTAVAIL:
                                syslog(LOG_WARNING,
                                    "error: EADDRNOTAVAIL");
                                break;
                        case EADDRINUSE:
                                syslog(LOG_WARNING,
                                    "error: EADDRINUSE");
                                break;
                        case EACCES:
                                syslog(LOG_WARNING, "error: EACCES");
                                break;
                        case EFAULT:
                                syslog(LOG_WARNING, "error: EFAULT");
                                break;
                        default:
                                break;
                    } //switch

                exit_failure("connect: %s", strerror(errno));
                /*NOTREACHED*/
        } //end if

        tcp_relay(s_src, s_dst, service);

} //end play_child

/*********************************************************************
* MRO
* Function:       fwd_choice
* Description:    Ensures the connection is from the client's known
*                 IPv6 address.
*********************************************************************/
/* 0: don't forward, 1: forward */
static int
fwd_choice(struct sockaddr_in6 *src)
{
        struct sockaddr_in6 *my6;
        char addrbuf[NI_MAXHOST];

        /*
         * if data is from client, then forward it to the server,
         * else do not service it.
        */
        getnameinfo((struct sockaddr *)src,
                src->sin6_len, addrbuf,
                sizeof(addrbuf), NULL, 0,
                NI_NUMERICHOST);

        if (!strcmp(clientAddr, addrbuf)) { /* if equal */
                if (!subSrcAddr(src)) {
                        syslog(LOG_INFO, "subSrcAddr failed!");
                        return 0;
                    } //end if

                return 1;
        }
```

```
	/* MRO: should never reach here, but default is fail */
	return 0;

} //end fwd_choice




/*********************************************************************
* MRO
* Function:	subSrcAddr
* Description:	Copies, perhaps unnecessarily, it's own address into
*		the source address field of the connected socket
*********************************************************************/
/* 0: fail, 1: success */
static int
subSrcAddr(struct sockaddr_in6 *source)
{
	char srcbuf[NI_MAXHOST];

	memcpy(source, &mySock->tpe_s, mySock->tpe_s.sin6_len);

	getnameinfo((struct sockaddr *)source,
		source->sin6_len, srcbuf, sizeof(srcbuf),
		NULL, 0, NI_NUMERICHOST);
	syslog(LOG_INFO, "New source address is %s", srcbuf);

	if (IN6_ARE_ADDR_EQUAL(&source->sin6_addr,
		&mySock->tpe_s.sin6_addr)) {
		    syslog(LOG_INFO, "subSrc verified");
		    return 1;
	  } //end if

	return 0; /* fail */
}



/*********************************************************************
* MRO
* Function:	grab_myaddrs
* Description:	Simply walks through all local addresses and stores
*		the relevant ones.
*********************************************************************/
static void
grab_myaddrs()
{
	struct ifaddrs *ifap, *ifa;
	struct sockaddr_in6 *sin6, *mysin6;
	char tpe_s[NI_MAXHOST];
char tpe_c[NI_MAXHOST];

	/* get a list of all addresses */
	if (getifaddrs(&ifap) != 0) {
		exit_failure("getifaddrs");
		/*NOTREACHED*/
```

```
	}

	for (ifa = ifap; ifa; ifa = ifa->ifa_next) {

		switch (ifa->ifa_addr->sa_family) {
		case AF_INET:
			continue; /* ignore IPv4 addresses */
		case AF_INET6:
			break;
		default:
			continue;
		}

	      /* MRO: store my server side address and log it */
	      if (!strcmp(ifa->ifa_name, ifServ)) { /* if equal */
		    mysin6 = (struct sockaddr_in6 *)ifa->ifa_addr;

		    if (IN6_IS_ADDR_SITELOCAL(&mysin6->sin6_addr)) {
			  memcpy(&mySock->tpe_s, mysin6,
				mysin6->sin6_len);

			  getnameinfo((struct sockaddr *)&mySock->tpe_s,
				mySock->tpe_s.sin6_len, tpe_s,
				sizeof(tpe_s), NULL, 0,
				NI_NUMERICHOST);
			  syslog(LOG_INFO, "tpe_s address is %s", tpe_s);
		    }
	      }

	/* MRO: store my client side site-local address and log it */
	      if (!strcmp(ifa->ifa_name, ifClient)) {
		    mysin6 = (struct sockaddr_in6 *)ifa->ifa_addr;

		    if (IN6_IS_ADDR_SITELOCAL(&mysin6->sin6_addr)) {
			   memcpy(&mySock->tpe_c, mysin6,
					mysin6->sin6_len);

			  getnameinfo(ifa->ifa_addr,
				ifa->ifa_addr->sa_len,
				tpe_c,
				sizeof(tpe_c),
				NULL, 0, NI_NUMERICHOST);

			  syslog(LOG_INFO, "tpe_c address is %s",
				tpe_c);

	      } //end if
	      } //end if

	} //end for

	freeifaddrs(ifap);
} //end grab_myaddrs


/*****************************************************************
* MRO:  The functions below have not been modified from the original
```

```
* faithd program
****************************************************************************/


static void
sig_child(int sig)
{
      int status;
      pid_t pid;

      pid = wait3(&status, WNOHANG, (struct rusage *)0);
      if (pid && WEXITSTATUS(status))
            syslog(LOG_WARNING,  "child  %d  exit  status  0x%x",  pid,
status);
}

void
sig_terminate(int sig)
{
      free(mySock);

      syslog(LOG_INFO, "Terminating faith daemon");
      exit(EXIT_SUCCESS);
}

static void
start_daemon(void)
{
#ifdef SA_NOCLDWAIT
      struct sigaction sa;
#endif

      if (daemon(0, 0) == -1)
            exit_stderr("daemon: %s", strerror(errno));

#ifdef SA_NOCLDWAIT
      memset(&sa, 0, sizeof(sa));
      sa.sa_handler = sig_child;
      sa.sa_flags = SA_NOCLDWAIT;
      sigemptyset(&sa.sa_mask);
      sigaction(SIGCHLD, &sa, (struct sigaction *)0);
#else
      if (signal(SIGCHLD, sig_child) == SIG_ERR) {
            exit_failure("signal CHLD: %s", strerror(errno));
            /*NOTREACHED*/
      }
#endif

      if (signal(SIGTERM, sig_terminate) == SIG_ERR) {
            exit_failure("signal TERM: %s", strerror(errno));
            /*NOTREACHED*/
      }
}

static void
exit_stderr(const char *fmt, ...)
{
```

```
	va_list ap;
	char buf[BUFSIZ];

	va_start(ap, fmt);
	vsnprintf(buf, sizeof(buf), fmt, ap);
	va_end(ap);
	fprintf(stderr, "%s\n", buf);
	exit(EXIT_FAILURE);
}

void
exit_failure(const char *fmt, ...)
{
	va_list ap;
	char buf[BUFSIZ];

	va_start(ap, fmt);
	vsnprintf(buf, sizeof(buf), fmt, ap);
	va_end(ap);
	syslog(LOG_ERR, "%s", buf);
	exit(EXIT_FAILURE);
}

void
exit_success(const char *fmt, ...)
{
	va_list ap;
	char buf[BUFSIZ];

	va_start(ap, fmt);
	vsnprintf(buf, sizeof(buf), fmt, ap);
	va_end(ap);
	syslog(LOG_INFO, "%s", buf);
	exit(EXIT_SUCCESS);
}

static void
update_myaddrs()
{
	char msg[BUFSIZ];
	int len;
	struct rt_msghdr *rtm;

	len = read(sockfd, msg, sizeof(msg));
	if (len < 0) {
		syslog(LOG_ERR, "read(PF_ROUTE) failed");
		return;
	}
	rtm = (struct rt_msghdr *)msg;
	if (len < 4 || len < rtm->rtm_msglen) {
		syslog(LOG_ERR, "read(PF_ROUTE) short read");
		return;
	}
	if (rtm->rtm_version != RTM_VERSION) {
		syslog(LOG_ERR, "routing socket version mismatch");
		close(sockfd);
		sockfd = 0;
```

```
            return;
      }
      switch (rtm->rtm_type) {
      case RTM_NEWADDR:
      case RTM_DELADDR:
      case RTM_IFINFO:
            break;
      default:
            return;
      }
      /* XXX more filters here? */

      syslog(LOG_INFO, "update interface address list");
      grab_myaddrs();
}

static void
usage()
{
      fprintf(stderr, "usage: %s [-dp] [-f conf] service [serverpath
[serverargs]]\n", myseatrname);

      exit(0);
}
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Dr. Cynthia Irvine
   Naval Postgraduate School
   Monterey, CA

4. Thuy D. Nguyen
   Naval Postgraduate School
   Monterey, CA

5. Joanne Kim
   Naval Postgraduate School
   Monterey, CA

6. Timothy Levin
   Naval Postgraduate School
   Monterey, CA

7. David Shifflett
   Naval Postgraduate School
   Monterey, CA

8. Richard Harkins
   Naval Postgraduate School
   Monterey, CA

9. Dr. Ernest McDuffie
   National Science Foundation
   Arlington, VA

10. RADM Zelebor
    N6/Deputy DON CIO
    Arlington, VA

11. Russel Jones
    N641
    Arlington, VA

12. David Wirth
N641
Arlington, VA

13. CAPT Sheila McCoy
Headquarters U.S. Navy
Arlington, VA

14. CAPT Robert Zellmann
CNO Staff N614
Arlington, VA

15. Dr. Ralph Wachter
ONR
Arlington, VA

16. Dr. Frank Deckelman
ONR
Arlington, VA

17. Richard Hale
DISA
Falls Church, VA

18. George Bieber
OSD
Washington, DC

19. Deborah Cooper
DC Associates, LLC
Roslyn, VA

20. David Ladd
Microsoft
Redmond, WA

21. Marshall Potter
Federal Aviation Administration
Washington, DC

22. Ernest Lucier
Federal Aviation Administration
Washington, DC

23. Keith Schwalm
DHS
Washington, DC

24. RADM Joseph Burns
Fort George Meade, MD

25. Howard Andrews
CFFC
Norfolk, VA

26. Steve LaFountain
NSA
Fort Meade, MD

27. Penny Lehtola
NSA
Fort Meade, MD

28. ENS Matthew O'Neal
Student, Naval Postgraduate School
Monterey, CA